internosis® perspective on:

.NET AND THE ENTERPRISE

By **Richard L. Warren, Chief Software Architect, January 3, 2003**

OVERVIEW

Microsoft has launched a new architecture for not only its operating systems, products, and solutions but a new way of connecting computer users, applications, systems, and companies that Internosis believes will have profound, far-reaching and unavoidable implications for enterprises of every size.[1]

If you use Windows for either servers or desktops, using .NET is not an issue – at issue is whether you'll plan it or not.

What many executives have neither recognized nor come to terms with is how little control they will have over whether their companies will adopt .NET. For the vast majority of companies the choices are functionally reduced to matters of degree and deliberateness – adoption of .NET in an ad hoc or intentionally planned manner.

Another aspect of the coming Web Services revolution (the approach and technology underlying Microsoft's .NET strategy) that has not been fully valued or anticipated is its ubiquity – the degree to which it will affect all business processes and their underlying information technology systems.

In this Internosis Perspective we begin by defining what .NET is from a non-technical, executive point-of-view. Next we examine what the business implications of .NET are along with the kinds of business drivers that will affect the speed at which .NET adoptions within enterprises will occur.

Finally, we look at strategies that not only come to grips with the competitive aspects of .NET but which also deal with corporate governance in the face of what promises to become the most disruptive technological force since the Internet itself.

---

[1] For purposes of this perspective Internosis is limiting itself to TCP/IP-based networks.

.NET – An Architectural Perspective

Most of the .NET "story" that has been positioned in the media from its introductory announcement to only recently has been technically-oriented and developer-focused. While that approach ignited incredible IT staff interest it has left business executives scratching their heads and asking themselves (and Internosis) three questions: (1) so what exactly is .NET; (2) what business problems does it solve; and (3) how much is it going to cost?"

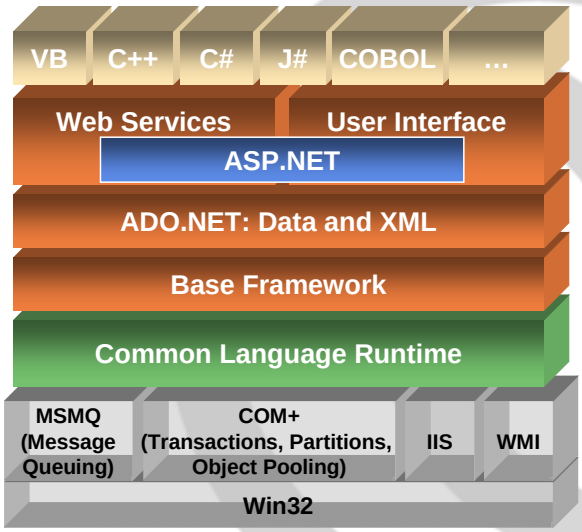
The "What" of .NET

The best way to begin to answer the first question is to explain what .NET is **not**: it's not a product. While there are several products with ".NET" in their name Microsoft has realized that doing so created more confusion than clarity in the marketplace. As a result Windows .NET Server 2003 is believed to be the *last* Microsoft product to have .NET as part of its name. [Note: Microsoft changed their .NET naming a week after this paper was released.]

Simply put, *.NET is Microsoft's strategy to put XML-based web services at the heart of everything they build or influence*. Key to understanding this strategy is what "XML-based" and "web services" means from Microsoft's perspective and how they are implementing them through the .NET Framework.

.NET is a strategy to make XML-based web services the cornerstone of the entire Microsoft architecture

At a conceptual level the .NET Framework is a "programming model" of the .NET environment that Microsoft has built into its Windows operating systems family. It includes everything from the languages programmers will need to build Windows and Web Services programs to the underlying "runtime" engine that allows Windows clients and servers to run .NET applications. As shown below, there are a number of layers in the framework. The top layer provides language



The Microsoft .NET Framework

support for the programmer. There are currently over twenty-two different language compilers that have been announced by Microsoft and their partners including COBOL and, surprisingly enough, the Java language.

Depending on whether the user of a .NET application is a person or another computer either the User Interface or the Web Services Interface is used. Data access and the underlying abstraction of the .NET environment are provided so that developers can program the same way for all kinds of applications and reuse components developed for one application within another even if the "user" changes from human to machine (or vice-versa).

The Common Language Runtime is where the real magic in .NET happens

Visual Studio .NET, Microsoft's new integrated development environment for .NET produces a common "intermediate language" file. This is the point at which real magic happens: components developed in different languages can all coexist within the same application because the Common Language Runtime (CLR) understands a standard set of data types and can pass information from one application component to another even if they were written in languages as different as COBOL and Java. And it is the CLR that ultimately produces the actual bits that run on the application server that sits below the entire .NET environment.

Without getting overly technical, Microsoft has traditionally created proprietary protocols and methods for communicating between programs or components of programs. The Component Object Model (COM) and its "easier to use" version (COM+) are the classic cases-in-point for this approach. Microsoft's first steps towards a standards-based approach in this area became known as the Distributed Component Object Model or DCOM. This protocol was designed to be used across a variety of network transports based on the Open Software Foundation's Distributed Computing Environment – Remote Procedure Call specification which permitted Microsoft applications to use both Java applets and ActiveX® components through its use of COM. While difficult to use these "binary protocols" were very effective (if inefficient[2]) as long as they were used within and across Microsoft-based networks.

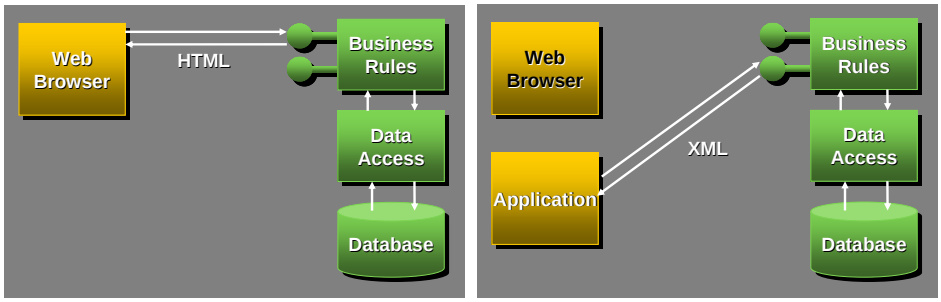
Once the Internet and Internet-based protocols and standards became widely used for both public and internal enterprise networks the fate of DCOM and its underlying COM architecture were sealed because, while DCOM could be

[2] Almost a dozen 'round trips' between a server and calling application were needed for each and every method call.

made to work through some firewalls[3] it wouldn't work through *all* firewalls, couldn't be made to work reliably, and couldn't be made to support the full range of functionality is was designed to deliver.

At the same time the HyperText Markup Language (HTML) was maturing as the basis for web site content description another Standard Generalized Markup Language[4] (SGML) derived language standard was developing to enable generic SGML to be funneled through the Web similar to HTML: the eXtensible Markup Language, or XML. Since its specification in 1998[5], XML and several related protocols have become the emerging standards for what are now being described as "web services": essentially, web sites that are meant to be used not by people, but by other computers and processes as shown below.

Web Browsing & Web Services



Just as you and I would use a search engine to find something on the Web, Web Services use a process called UDDI, Universal Description, Discovery, and Integration to find services and WSDL, Web Services Description Language, to figure out how to use them.

Finally, we use a protocol known as SOAP, the Simple Object Access Protocol, to get these services to actually do something or return information we need.

By combining these simple building blocks Microsoft has set the stage for the evolution of information technology by allowing almost any kind of system to connect to almost anything else.

The ubiquity and interoperability of the web applied to data instead of content

What this approach offers is, for the first time, the same ubiquity and agnosticism as exists in the World Wide Web in general. Services can now be discovered, described, vended, and utilized using the same techniques and network topologies that were developed for the Web whether those services exist on

[3] DCOM dynamically opened one port per process and required both UDP and TCP ports 135-139 to be open. While network engineers could alternatively build a TCP tunnel neither approach would support callbacks through the firewall.
[4] ISO Standard 8879 of 1986
[5] W3C Recommendation 19980210 (available online at http://www.w3.org/TR/1998/REC-xml-19980210)

the opposite side of the globe or within the most protected subnets of private networks. This permits systems and processes to be integrated in ways which were never before possible with DCOM, OMG's CORBA or any other distributed object system because they can be more "loosely coupled," emulating the way a browser is coupled to a web server (we'll discuss why "loose coupling" is so important in the next section – suffice it to say for the moment that it is *crucial*).

To implement their .NET strategy Microsoft has created both a framework (described above) and delivered the supporting developer tools needed to build web services upon it. For versions of its operating systems prior to the forthcoming Windows .NET Server 2003, this framework is realized by adding it onto the existing OS while, with the release of Windows .NET Server 2003, the framework will be part of the operating system itself.

.NET is the most profound shift in Microsoft architecture in almost ten years

To build web services capabilities into all of its application servers (e.g. Exchange, SQL Server, Commerce Server, etc.) Microsoft has taken a three-phased approach. Phase one involves adding a fairly superficial 'wrapper' to their existing product line that acts as an XML facilitator to the underlying COM architecture. Phase two moves the XML-based web services closer to the core of the products and provides a much richer interface for developers to begin using them as the source of specialized web services (e.g. Commerce Server 2002, Content Management Server 2002, etc.). Phase three will involve completely changing the underlying architecture of applications so they will communicate using only XML-based web services. Internosis believes this third and final phase will be the most significant shift in Microsoft's entire architecture since the introduction of Windows NT 3.1 in July, 1993.

The "Why" of .NET

Why Microsoft would want or need to create a .NET strategy and radically change the architecture of their entire product line is tied up in the second, more important question, the answer to which is, "The problems that cause your business to react too slowly to rapidly changing competitive situations or react so slowly you can't effectively raise barriers to competition."

Integration complexity compounded by system diversity and multiple solution techniques translates directly into cost multipliers

The problems .NET solves in a very real but general sense involve information systems that support your line-of-business operations that are either so tightly integrated or 'coupled' that your IT team can't quickly enough change anything without breaking everything or so totally isolated they stand as 'information silos' that aren't integrated with anything else. It gets worse...

As bad as most enterprise 'coupling' problems are they are typically compounded by having to use different techniques to solve the same problem depending on whether the problem exists between LOB information and a corporate user, that same information and another system within your enterprise, or that very same information and a system in one of your supply- or demand-chain partners. This directly translates into headcount, cost, redundancy, and hopelessly interlocked (and deadlocked) project schedules.

Microsoft has been trying over the last year to position their .NET message in terms of "agility." You may have seen television ads showing a totally integrated supply- and demand-chain system using everything from legacy systems to handheld PDA's that "didn't exist when I first got here…a couple of months ago." While nobody we know is apt to believe the suggested timeframe in that ad, everybody we know wants their IT organizations to be more responsive to the kinds of business needs described above. Gartner has picked up on this theme as well:

> In the past, enterprises took what AD [application development] organizations delivered, albeit with some complaining about software quality, low productivity, delays, high costs and project cancellations. AD organizations have survived because enterprises needed, and were willing to accept, whatever the AD organizations delivered, whenever they delivered it.
>
> This is no longer true. Enterprises in the era of e-business simply can't afford to wait years for AD organizations to build applications; however, 'traditional' AD organizations aren't built to do things differently.[6]

Gartner prediction: Application development groups will exist on only one-third of enterprise IS teams by 2005

The problem has become so bad that Gartner predicts that by 2005 traditional application development groups will exist in only one-third of all IS organizations (60% probability).[7]

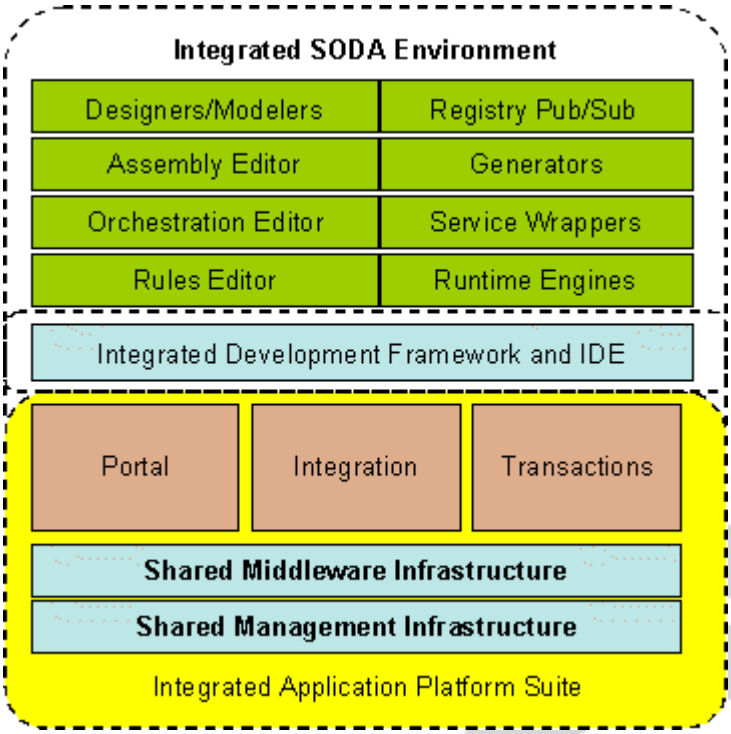
The emerging trend Gartner has observed they are calling the Service-Oriented Development of Applications, or SODA. While Internosis doesn't have as memorable an acronym the resulting integrated environment for SODA they see bears a striking resemblance to the .NET framework currently available with first- and second-phase .NET servers available for all blocks of the diagram.

---

[6] Light, Duggan, *Pressure on Application Delivery: Something's Got to Give*, AV-18-9403 dated December 4, 2002, page 1, Copyright © 2002 by Gartner Research (accessed online on December 27, 2002)
[7] IBID, page 1

As important as the similarity between Gartner's analysis and Microsoft's .NET Framework is their assessment that for Microsoft to **not** be a market leader through 2006 along with IBM would require the current COM/DCOM customer base to reject .NET or for Microsoft to fail to enhance its enterprise scalability sufficiently, neither of which is apparently the case given the .NET adoption statistics and case studies available and the scalability Internosis has observed in the forthcoming Windows .NET Server 2003 family of server operating systems. The scenario is deemed so unlikely Gartner has rated the possibility "negligible" with the recommendation that, "Risk-averse enterprises looking to pick a long-term leader should select either IBM or Microsoft (or both if they want to support J2EE and .NET).[8]

The Relationship of ISE and APS
[Integrated SODA Environment and Application Platform Suite]



Source: Gartner Research[9]

Internosis has since its inception focused on reducing the risks associated with technology change management while simultaneously working to reduce the attendant decline in productivity. While this approach doesn't guarantee the

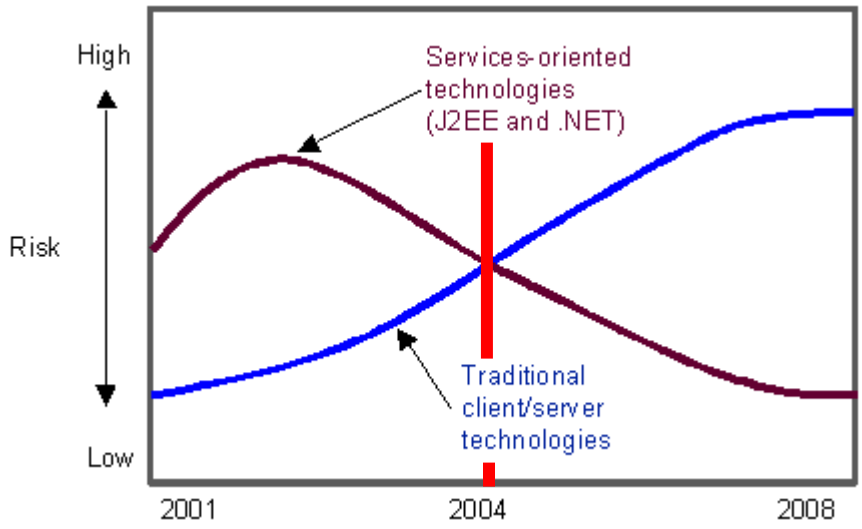
---

[8] Blechar, Duggan, Jones, Hotle, *Some Vendors Will Survive SODA, and Others Won't*, Report SPA-18-7422 dated November 25, 2002, page 7, Copyright © 2002 by Gartner Research (accessed online on December 27, 2002)
[9] IBID, page 2

breakeven point of any given project moving to the left on the timeline (earlier recognition of net positive impact), it does directly and positively impact ROI.

What Gartner foresees may so increase the risks of technology change management by 2004 that no amount of risk management will have a positive impact. As shown below the changing technology risk associated with both .NET and J2EE is believed to have peaked in 2002 and is now in constant decline through 2007. Simultaneously the risk associated with "traditional" client/server-based technologies have been increasing and will continue to do so through 2007 "breaking-even" with the .NET/J2EE risks by 2004.[10]

Changing Technology Risk



Source: Gartner Research[11]

The Cost of .NET (and Higher Costs of NOT .NET)

While there is no acquisition cost for Microsoft's .NET Framework add-on for the Windows 2000 Server family or any of the Windows clients per se there are much larger total-cost-of-ownership issues afoot and a number of business drivers that will affect both the adoption curve and useful investment recovery timeframes. This makes the answer to, "How much does it cost?" more an approximation of the risks and rewards than an estimate of out-of-pocket expense.

---

[10] Blechar, *SODA Requires People, Process and Paradigm Shifts*, Research Note SPA-18-7280 dated November 25, 2002, page 1, Copyright © 2002 by Gartner Research (accessed online on December 27, 2002)
[11] IBID, page 1

Optimum transition to service-oriented platforms will conclude by 2004

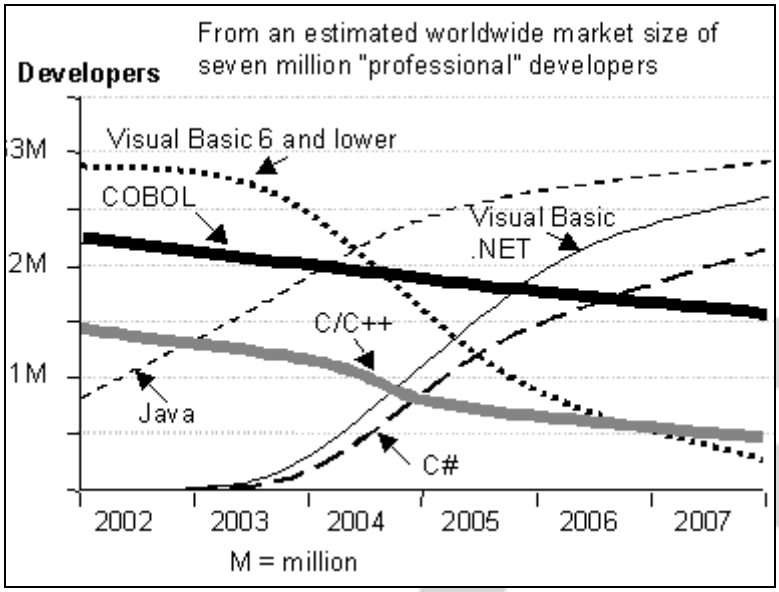
Looking again at Gartner's Changing Technology Risk figure above it is fairly obvious that (1) any organization that has not transitioned from traditional client/server technology development by 2004 will face increased risks, and (2) that those risks will be greater by 2006 than they were in 2002 at the height of the risk curve for .NET and J2EE.

The factors driving those risk curves are several and varied:

Staffing risks

- The evangelists of both J2EE and .NET have been focusing most of their efforts on developers, who are increasingly beginning to update their skill sets. Enterprises will increasingly face ***staffing risk*** for "legacy client/server applications" as the developer workforce becomes predominately service- and object-oriented. As shown below, Gartner shows a steady decline in legacy language workforce (COBOL), precipitous decline in the client/server languages (VB6 and C – particularly in 2004), and dramatic growth in the service-oriented languages (C#, VB.NET, and Java – although there is a considerable shouldering of that curve in the 2004/2005 timeframe).

Number of Professional Developers Worldwide



Source: Gartner Research[12]

The decline in both legacy and client/server development staffing will drive an increase in off-shore development for those organizations that fail to transition to services-oriented application development environments such as .NET and J2EE, which addresses the staffing risk while

[12] Felman, Driver, *Leading Programming Languages for IT Portfolio Planning*, Research Note SPA-17-6636 dated September 27, 2002, page 3, Copyright © 2002 by Gartner Research

creating a whole host of additional risks not the least of which are additional security risks.

- Because the leading vendors have been and are continuing to target developers (discussed above) and because developers are urgently interested in keeping their skills and résumés up-to-date (with developer's being under employment pressure for the first time in over a decade) there is a substantial risk to enterprises that these AD platforms and technologies will enter use on an uncontrolled, unplanned basis: a de facto "viral" adoption.

"Viral adoption" risk

- Support for the tool sets that facilitate client/server development have already begun to decline as vendors shift their focus. Enterprises will increasingly face ***developer tool vendor support, update/bug-fix, and maintainability risks*** as the vendor base for C/S tools both diminishes and as the tool vendor market consolidates.

  Beyond the tool set vendor support issue is the larger issue of Microsoft's support for pre-.NET technologies in the future. Microsoft has not been known for its long-term support of no longer used technology as, by contrast, has IBM. While Redmond has more clearly enunciated their policy on product end-of-life support and has made significant concessions in the face of customer feedback, there is the very real risk that users of Microsoft-based technologies that do not adopt .NET will find themselves in an undesirable support position three to five years hence.

Support risk – Third-party and Microsoft itself

- Training costs associated with .NET and J2EE platform transformation, which have been typically high during the 'early adopter' phase (to recover curriculum development costs), will hit 'mainstream' cost points during 2003 then rise again as demand diminishes. Enterprises will increasingly face ***expense risk*** to retrain their application development teams if they miss the imminent, optimum cost point.

Training expense risk

- Client/Server application development relied upon a relatively stable requirements base owing to the average project duration being measured in months (if not years). Current business competitiveness demands have effectively reduced the "requirements half-life" to only months, reducing the cycle-time for application development to durations of only weeks. Enterprises will face substantial ***application re-development risks*** if they either (1) superficially transition from C/S development without actually changing the underlying application development

Re-development risk

processes or (2) fail to implement governance processes that will segment and differentiate invocators/users of services from the underlying services functionally and business rules through the use of flow management and object reuse facilitators.

- Just as application development tool vendors will shift their focus from C/S to service-oriented platforms so too will (and have already) operating systems vendors. In a relatively short timeframe this will mean that enterprises will face increased ***security risks*** as these vendors focus on emerging threats to and fixes for their service-oriented platforms as their highest priority, leaving "legacy" client/server environments more at risk through less responsive and slower security support. Recent press reports indicate companies with security support issues are already beginning to get "resolution through upgrade" answers much to their chagrin.

Security risk

Just as with the above risks, the enumeration of the benefits side of the equation requires a foundation too. But gathering hard comparative data on a framework only a year old and (worse) an operating system that is still unreleased provides a number of challenges.

The only head-to-head comparisons where an independent, third-party testing laboratory had tested and verified the results were the J2EE reference and best-practices application, the "Java Pet Store Sample Application v1.3.1," and the "Nile @Bench" benchmark (although the IBM Websphere® license prohibits publishing benchmarking results).

Based on Veritest's comparison[13] (the results of which are shown below) on identical hardware enterprises should expect both substantial performance and scalability advances from the .NET platform.

- Source: Lionbridge Technologies, Inc.[14]

While raw application performance will have a significantly front-loaded impact on project costs, it is the life-cycle total-cost-of-ownership (TCO) that gives the truest picture of .NET's potential impact. ***From Internosis' perspective there is simply too little information yet available to form a definitive viewpoint.***

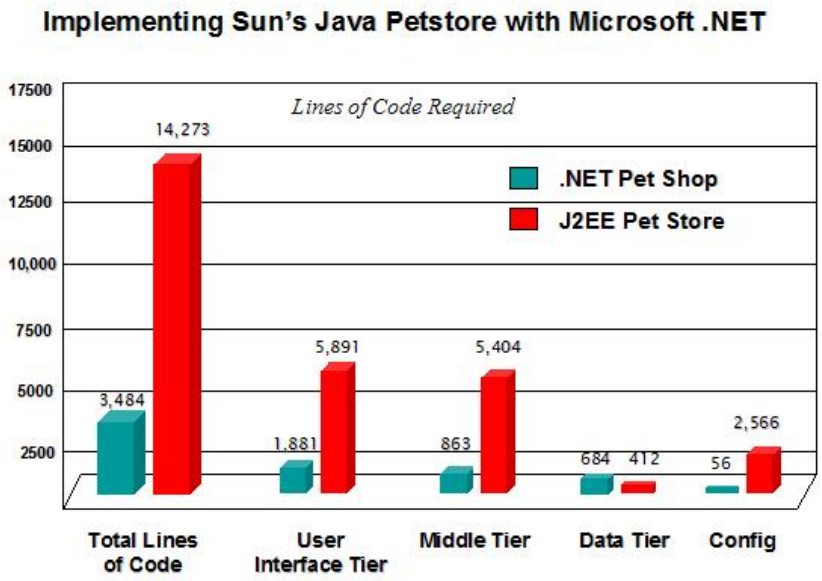
What information is available points to a ***probably favorable*** TCO based on the following:

---

[13] *VeriTest® Benchmark and Performance Testing Microsoft® .NET Pet Shop, May 2002*, Copyright © 2002 by Lionbridge Technologies, Inc. [accessed online January 5, 2003 at http://www.gotdotnet.com/team/compare/Benchmark_ShortRepFinal.pdf]

[14] IBID

- **Developer Productivity –** The same reference application cited above, the Pet Store application, provides a useful insight into how much more developer productivity may be possible with .NET by looking at the longtime standard for project scope and complexity: lines-of-code. As shown below, the total lines of code needed to achieve identical functionality is approximately 25% of that needed in the closest SODA architecture, J2EE.



Source: Direct comparison of the source code files provided by Microsoft and Sun Microsystems[15]

This differential should impute large TCO advantages if the same (or substantially similar) percentages hold on a more general basis. But, reiterating our previous caution, in our estimation there is still too little audited evidence available to form definitive conclusions.

Internosis *can* convey initial impressions formed by our own developers and those in the application development communities in which they interact, however. Almost to a developer, none would willingly return to a pre-.NET development environment after learning and having used it. Moreover, the general impression is it takes approximately 50% less time to accomplish the same set of tasks using the .NET Framework and

[15] Microsoft source code available from: http://www.gotdotnet.com/team/compare/codepetshop.aspx. Sun Microsystems source code available from: http://java.sun.com/blueprints/code/index.html#java_pet_store_demo.

tools. *They have found the key to success is in learning and leveraging the large base class structure and functionality that is the heart of the .NET Framework.*

- **Granularity –** Applications that were once more-or-less monolithic and so tightly-coupled that nothing could be changed without everything being changed should be less costly to maintain over the life-cycle because the loosely-coupled nature of .NET Framework applications will allow individual modules, presumptively based on clearly defined functionality, to be optimized or replaced during the life-cycle at a substantially reduced cost/complexity.
- **Adaptability –** Arising out of modular reuse, .NET promises the same (or better) adaptability as has already been observed in Java/J2EE application development environments. While neither .NET nor Java technologies can deliver on the promise or reusability (no technology by itself can - only enterprise processes, best practices, and both broad and deep object library knowledge can truly deliver on that promise), .NET places the Microsoft-centric AD environment on a par with its J2EE counterpart through the framework base class structures with performance superior to J2EE because it *does* take advantage of the specific underlying operating system at the expense of interoperability.
- **Reliability –** Internosis refers here not to the hardware or infrastructure, but to the process for application development itself. Project complexity has always escalated risk and usually by more than linear factors. By providing a truly comprehensive object framework for the Windows operating system environment, .NET should facilitate project decomposition into functional modules (classes) which through loose-coupling will be more easily unit-testable and isolated from change management complexity issues that have traditionally driven up life-cycle costs. Constraining the impact of change to only the modules/classes that provide the underlying functionality should have a large, positive impact on TCO through reduced risk and cost-of-failure (change small, fail small…as described below).
- **Innovation/Reduced Risk Aversion –** Another impact Internosis expects to see on TCO from .NET implementations is heightened application development innovation or, viewing the same coin from the opposite side, reduced aversion to risk. Because a more granular application development environment tends to improve reliability (as discussed above), project failure risk should decline as newer applications

become more and more adaptations of previously developed, tested, and reliable applications – building on a trusted code/object base reduces the "risk surface" which should, in turn, reduce enterprise adversity to taking such risks because they will be smaller, more susceptible to remediation and granular enough that setbacks in individual modules/classes are less likely to resonate throughout a project with catastrophic effects.

- **Integration Ubiquity –** Many enterprises with which Internosis works have expressed concern over the large base of still business-critical legacy code and the life-cycle cost of migrating it (or redeveloping entire application suites) to keep those applications running on supportable platforms. While there have not yet been sufficient references made available publicly to form a convincing conclusion Internosis is none-the-less encouraged by the potential for .NET to provide both "wrapper" and lower cost redevelopment strategies for sustaining and supporting the usefulness of these legacy applications – particularly where the underlying AD language is COBOL. Applications written in COBOL and running on a variety of pre-Internet systems can, in many cases, be easily "wrapped" with high-performance Web Services to expose their underlying functionality.

  In those cases where cost-efficiency or impending system replacement makes application redevelopment an alternative, .NET support for many legacy and niche languages with standardized data typing between modules has introduced a much lower cost opportunity. While many CICS environment functions will still need to be redeveloped, the support of COBOL as a full .NET language suggests that a large percentage of the existing, most complex legacy code – the code which provides business logic – may be ported in a relatively straightforward manner.

- **Security –** Within the Windows environment the reduction in the kind and number of application-to-application, system-to-system, and business-to-business interfaces and protocols to a set of fewer, better standardized, more widely understood, and easily (and better) monitored interactions should substantially reduce the threat surface.

  At the same time, the reduction in the number of ways developers invoke system services and standardization of those invocations through both vendor-provided base classes and more modular and testable extensions to them should provide better identification of impacted appli-

cations when vulnerabilities to specific system calls are identified. Moreover, because of their highly modular nature, the time-to-fix for individual vulnerabilities is likely to be shorter for the reasons discussed above in the Reliability discussion.

- **Velocity –** The net/net of these favorable factors should be that response to rapidly changing business needs should be quicker, have a lower TCO, and be more secure owing to the combination and interaction of the foregoing. ***This will have the effect of increasing net enterprise competitive velocity.***

OBSERVATIONS & RECOMMENDATIONS

Observations

Internosis has observed one trend and two developing scenarios that could substantially impact both enterprises' decisions *whether* to implement .NET and the *speed* at which such adoptions may occur.

Center-of-Gravity Shift

While most successful IT strategies are multi-faceted, our experience indicates exceptionally successful (and popular) strategies have a particular emphasis on the 'back office,' the mid-tier, or the desktop. This emphasis forms a kind of center of organizational "gravity" which regulates and orders investment and focus in the other two areas.

The shift many industry pundits expected from PC's with rich user experiences to thin clients with essentially browser-based experiences did not take hold and is unlikely to going forward. At the same time device diversity has and continues to increase as PDA's, web-enabled cellular phones, and even Tablet PC's extend the enterprise network beyond the physical LAN/WAN.

The net effect for most enterprises will be a shift in the "center-of-gravity" toward the desktop and *whatever it takes to produce the most productive, cost-efficient experience there.* This shift is likely to have at least two noticeable side effects:

1. Application development organizations within enterprises will retool and focus even more closely on the tools and environments that are most effective at development for whatever operating system and hardware architecture dominates their organizational desktops – largely Windows; and

2. Application development organizations within enterprises will retool and focus on ubiquitous interfaces to back end systems that can be easily consumed by those desktop and mobile applications and external business partners.

Given the predominance of Windows on the desktop, Internosis believes this shift will favor and accelerate the adoption of .NET since the .NET framework can be applied to "legacy" Windows versions.

Java.NET?

Java for .NET

Microsoft already lists Java as a .NET language although no publicly announced third party has released a Visual Studio .NET version of that language. Internosis has been told that Rational Software has developed a "from the ground up" implementation of Java for .NET. If confirmed, and given the announced acquisition of Rational by IBM, this could provide complete language neutrality within the existing .NET environment with no further changes required by Microsoft for Java language developers to immediately and completely leverage that environment where it makes sense for them. Since it would also be consistent with IBM's other investments in Java to support their broader Linux initiatives, Internosis observes this is a completely consistent and plausible development which would have significant enterprise impact on .NET adoption.

Linux.NET?

Open-Source for .NET

Just as Java for .NET would have an effect on .NET adoption from the development side, being able to run .NET Common Language Runtime code on a non-Windows platform would also have very significant impact on .NET adoption and velocity.

Internosis observes an Open-Source Software (OSS) version of the .NET Framework for non-Microsoft platforms (specifically Linux) would strongly tip the decision factors in favor of .NET irrespective of the underlying server or client technology. Several OSS organizations have announced their intention to do just that and have open projects at this time. They have announced they intend to have a version of their framework available during 2003.

Recommendations

Achieve more responsive business-value realization

Internosis recommends you create (or update) a technology change management strategy that focuses on higher efficiency, less code-centric, service-oriented application development and higher, more responsive business-value realization.

Within this context, "more responsive" directly translates to "velocity:" the speed at which IT organizations can support business requirements. Gartner Research predicts, "by 2007, the average duration of a project that deploys software will be less than four months (0.6 probability). As recently as 1997, this duration was measured in years. Today it's measured in months. In the future, it must be measured in weeks."[16] The pseudo Rapid Application Development (RAD) environments and methodologies in use today typically only produce bad software faster. What is required to overcome the velocity/quality mismatch is a services-oriented application development environment built upon web services and leveraging component reuse.

What is required is a services-oriented AD environment built upon web services and leveraging component reuse

- If your enterprise (1) predominantly uses Windows at the desktop, (2) predominantly uses Windows 2000 Server or Windows NT 4 at the mid-tier or back office (or your mid-tier or back office mix and application development trend strongly favors Windows 2000), or (3) lacks predominant Java development expertise, Internosis recommends you begin immediately to plan for eventual .NET adoption as a way of implementing this recommendation.

Windows-predominant enterprises should plan for and adopt .NET sooner rather than later

- If, on the other hand, you (1) predominantly use Linux or Unix at the desktop, (2) predominantly use Solaris or Linux at the mid-tier or back office, or (3) have predominantly Java development expertise, Internosis recommends you begin immediately to plan for a loosely-coupled, services-oriented architecture

Linux/Java-predominant enterprises should plan for and adopt services-oriented architectures as soon as they are supported

- If your enterprise needs to have both Microsoft-centric and Java-based application development capabilities Internosis recommends you begin immediately to plan for eventual .NET adoption for the Microsoft-centric portion of your needs and closely monitor Java .NET and OSS .NET Framework initiatives with an eye toward eventual application development language unification.

Heterogeneous environments should unify their analysis and requirements process and implement both .NET and J2EE

Implementation steps:

a. Control and plan for the introduction of new application development methods, tools, and platforms.

Control .NET introduction

   As mentioned in the .NET Risk section, introduction of .NET was originally targeted at developers not CIO's. This "bottom up" approach can have an incredibly viral effect upon enterprise IT organizations.

---

[16] Hotle, *Commentary: AD Cultural Revolution – Services-Oriented Development*, Research Note COM-18-9221 dated November 27, 2002, page 2, Copyright © 2002 Gartner Research.

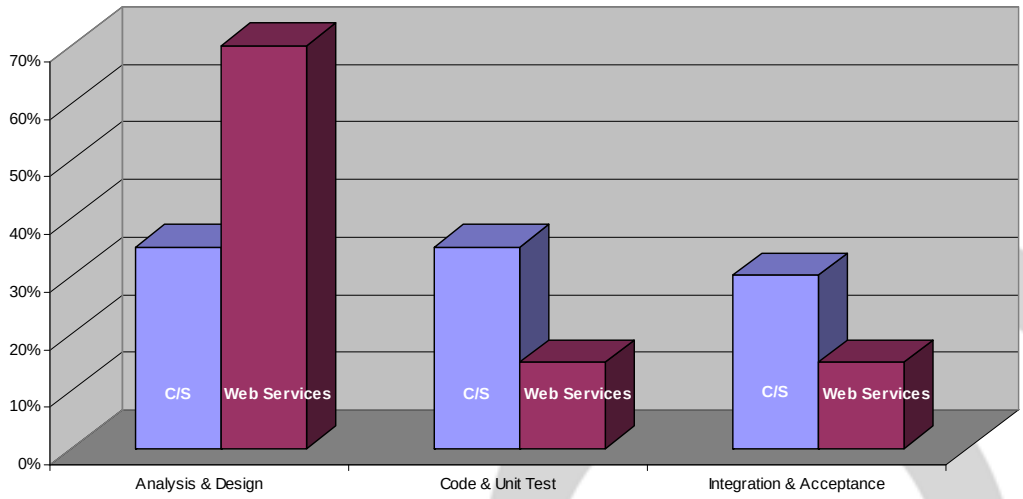
Implement the "best defense is a good offense" tactic by piloting a .NET project where it makes sense within your enterprise. It will function as a 'magnet' to developers interested in .NET and simultaneously allow your IT management team to focus their efforts and growth in ways which can best suit broader .NET eventualities and cost/benefit metrics gathering.

b. Choose one application development process/methodology and at most two platforms.

Unify front-end requirements and analysis processes

Shifting from client/server to a services-oriented application development environment shifts ***where*** enterprises need their core assets. As shown below, the amount of effort in the up-front application development cycle for web services-oriented applications *doubles* while the coding, testing, integration, and acceptance of services-oriented project drop precipitously. ***This strongly argues for a unified requirements analysis and design methodology*** such as Rational's Requisite Pro coupled with their XDE add-in for the development tools that support both .NET and J2EE implementations (tailored to meet specific organizational needs).

Services Development Life Cycle Shifts



Source: Gartner Research[17]

It is the de-emphasis on coding that will allow enterprises to focus on the architectural shift to service-oriented development architectures

[17] Light, *Commentary: AD Project Management – Requirements Revisited*, Research Note COM-18-8738 dated November 26, 2002, page 2, Copyright © 2002 Gartner Research

without having to focus (fixate, really) on the "religious" aspects of language choice/migration issues.

c. Develop processes and implement tools that will facilitate code/component reuse.

Reuse within component architectures such as .NET and J2EE is the key

Gartner Research predicts that, "by 2007, more than 70 percent of the software that is newly developed into enterprises will contain services that were developed for use in other deployments."[18] That level of component reuse will require strong change management processes and flexible, extensible, and (most importantly) "queryable" object code repositories.

Most enterprises will initially choose an out-of-the-box product that meets most of their needs but which can eventually be extended to meet emerging requirements. For those enterprises choosing the .NET Framework, integration with Visual Studio .NET will be a "must" narrowing the choices primarily to either Microsoft's SourceSafe or market leader Rational's ClearCase (usually coupled with ClearQuest® for comprehensive software configuration management).

More important than the enterprise choice of supporting technology is the creation of new application development processes that will create a culture of component reuse.

As component reuse becomes the "norm" instead of the exception, enterprises can rise above them to implement patterns that are even now beginning to emerge within the services-oriented application development environment. But that's the next stage…

About Internosis

Internosis is an innovative information technology consultancy that ensures predictable outcomes as it delivers business-driven, Microsoft-centric solutions that increase agility and enable sustainable competitive advantage. Powered by the proven tactical model *Assured Performance*, Internosis is a Microsoft Gold Certified Partner for both Enterprise Systems and E-Commerce Solutions, a Pivotal Certified Systems Integrator, and a six-time Microsoft Certified Partner of the Year.

If maximizing ROI from IT investments is your driving concern, visit www.internosis.com or contact us at 800.274.8381 or info@internosis.com. Internosis is backed by General

[18] Op Cit, Hotle, page 1

Atlantic Partners, the world's leading private equity investment firm focused exclusively on global IT and communications investments.

This white paper was written by Richard L. Warren, Chief Software Architect.
He can be reached at info@internosis.com.