Infostructure Associates

Delivering TCO/ROI Via High-Level Development: Four J's Genero

White Paper

Infostructure Associates
an affiliate of Valley View Ventures
113 Bristol Road
Wellesley, MA 02481
Telephone: 508-265-1558
www.valleyviewventures.com

Table of Contents

Delivering TCO/ROI Via High-Level Development

Executive Summary

Increasing numbers of CIOs and CEOs are finally beginning to realize that *the choice of development solution and platform matters to the bottom line*.

Why now? The short answer is "***speed to value***." Today, as never before, IT and ISVs can cut the time to create a high-value application from 2 years (or never!) to 12 months, simply by choosing a different development solution. At the same time, the value of that additional 1-1 ½ years to the organization has skyrocketed, because computer applications are now a major source of competitive advantage, and competitors can copy that advantage in shorter and shorter periods of time. In other words, when product turnover is 3 years or less, being alone in the market for half that time is worth far more than in the old days, when IT mattered less to competitive advantage and long product lifecycles meant that competitive advantage lasted for less of the product's life span.

However, the short answer does injustice to the full sweep of the new trend. This is about speed to *value*, not just speed to market. Speed to value involves two other key criteria for the organization:

1. Cost
2. Risk

That is, an application may be valuable not merely because it gives competitive advantage, but also because it cuts costs, or mitigates corporate risk.

Obviously, an application can be aimed directly at cutting costs or implementing business continuity. However, it is also becoming increasingly clear that the same tools and infrastructure components that speed development also cut application cost of ownership (which is a significant corporate cost), and reduce the risks that the application will fail (which is a major factor in business continuity).

Not all development solutions deliver this value-add. Today's complex architectures make development using a simple toolset and programming language a lengthy, costly, and failure-ridden process. The key characteristics of development solutions that deliver long-term competitive advantage, cost, and risk-mitigation improvements are:

- *High-level programming and targeted "platforms"* of components that improve programmer productivity.
- *Support for leveraging existing software and for rapid upgrade* and customization that cuts development-project costs.

- *Flexibility* that allows rapid incorporation of new technology into existing applications, ensures that both the development toolset and the application integrate with the enterprise's architecture and other tools, and allows the toolset and application to adjust rapidly and easily to changes in the enterprise infrastructure.

In this *White Paper*, we show how the choice of the right development solution can lead to significant business benefits in per-application and overall TCO, ROI, competitive advantage, and risk mitigation. In particular, we examine Four J's Genero and Genero Studio development platform, a strong example of a solution that can deliver major benefits in programmer productivity, application flexibility, and application robustness, which, in turn, allow enterprises to achieve competitive advantage, cost savings, and lowering of risk.

The Real World: Genero Case Studies

Development Solution Case Study: Government ISV

This case study examines an ISV of ERP software (e.g., general ledger, accounts payable, payroll, and human resources), including handling tax billing, water/sewer billing, building permits, and code enforcement applications, for state and local governments. Typical target environments include Unix, Linux, and Windows for operating systems, and Oracle, Informix, and Microsoft SQL Server for back-end databases. Users' database sizes range up to 100 GB.

The ISV had used Four J's toolset before Four J's introduced Genero, and when Genero arrived, chose to "migrate" all of its products to use the Genero platform and toolset. Before choosing Genero, the ISV prototyped its next-generation product using a Microsoft toolset, but found that development "took longer" and would have resulted in an "extended and very costly migration." The ISV finds that both development and customization (especially screen generation) are faster with Genero than with Microsoft's Visual Basic. The resulting applications are "very scalable."

The ISV especially values Genero's "abstract user interface", which allows user sites to customize fields and screens for each site's town or city easily and rapidly. Administration of the resulting applications (typically only involving backup) takes "next to no time", and it is easy to administer sites remotely. The ISV acts as an ASP for 40 customers, and finds that only two administrators are needed for all administrative tasks. Training costs for developers are minimal (there is "not a huge learning curve"), and especially because of the object-oriented components that Genero provides for developer use. Deployment is very easy — "many tasks can be accomplished remotely." The level of support from Four J's is "much more than from Microsoft and Oracle."

The ISV ascribes clear, bottom-line benefits to improvements in its applications resulting from adoption of Genero — they have had "a great year in what is generally considered a soft market," due to the improved "look and feel" of the applications. In particular, customers are excited by their new ability to customize such features as field labels and

tool tips for themselves. The ISV estimates a 5-15% positive impact on revenues from the improvements.

Development Case Study: Latin America

This case study involves an in-house developer of social services software (e.g., information on health care and housing) that supplies information to 10 million people (2.1 million civil servants, dependents, and retirees) in a major Latin American country. The applications run on Sun Solaris servers using terabyte-scale IBM/Informix databases. The development organization uses Genero primarily for new-application development, including the Affiliation health-care program that provides clinical data, a credit application, and Instamed, a health-care application as "E-services" (Web services with Web user interfaces).

The development organization adopted Genero in the last two years, replacing mainframes running COBOL applications. The development organization finds that improvements in development time are especially clear for "maintenance" — it is "easy to implement changes." Moreover, the toolset makes it easier to improve application quality by following development-process standards such as CMM.

Compared to previous tools, the development organization finds Genero easier to learn — training is "much less complicated." Likewise, the resulting applications are easier to support (because the applications are "easier to use"). The applications scale to terabyte-sized databases and millions of end users "fine".

There is high customer satisfaction with the new applications — they have "revolutionized" the way that end users contact and interact with the government. Moreover, maintenance costs for the applications are lower, and the development organization is "able to meet deadlines [for delivering new applications] without hassle."

Analysis: Problems With Present Development Solutions

Recently, despite warnings, users have flocked to rudimentary Java tools, immature Web site authoring tools, and coding directly over a low-level Web-server or application-server interface, such as CGI, turning away from the high-level approaches of independent suppliers. The results have been:

1. A significant downturn in programmer productivity, measured in numbers of lines of assembler code per day.
2. Inflexible and inordinately complex "spaghetti" software that has caused excessively high application-upgrade and application-maintenance administrative costs.

These are not theoretical findings. A recent Infostructure Associates case study notes:

> "The application was developed [using a high-level development solution] in just under 2 years with an average of 3 programmers at a cost of less than $1 million. Had the supplier chosen Microsoft [C# or] Visual Basic .NET, development was estimated at 3 years, with a cost of $10 million; and those estimates were viewed as optimistic, because a competitor

attempting a similar application using Microsoft's tools had spent close to 6 years and $100 million for a less-scalable solution."

Problems with Object-Oriented Lower-Level Programming Languages

A high proportion of development projects continue to be late and over budget, while many large-enterprise IT shops continue to sport backlogs of a year or more, typically composed of projects that are needed. However, most attempts to "fix the problem" have ignored the positive role that the right development tools can play, or have opted for fads such as Java object-oriented lower-level programming languages without considering the side effects of the new technology. Table 1 shows the minimal programmer-productivity effects of object-oriented Web-application programming compared to some other programmer-productivity techniques.

Object-oriented programming using the C^{++}, Java, or $C^{\#}$ 3GLs creates object classes comprising code and the data it operates on, and these object classes interact during runtime in a loosely coupled, asynchronous fashion. Descriptive (higher-level) programming "describes" the operations of a program at a high level, allowing a program generator to create low-level object-oriented or procedural code. Java and C^{++} programming are object-oriented; Web page authoring, VPE (Visual Programming Environment) programming, and program designs are primarily descriptive, and Genero Studio programming offers descriptive features.

Table 1: The Programmer-Productivity Effects of Today's Techniques

Technique	Design	Coding	Testing/Deployment	Likely Maximum Time Saved
Generic development lifecycle (3GL)	25%	45%	20%/10%	(n.a.)
Object-oriented programming	30-35%	25-35%	20%/10%	10%-20%
Higher-level transactional coding	10%-20%	10%-30%	10%-15%/10%	40%-60%
Components/Frameworks	20%-30%	25%-35%	15%-20%/10%	5%-15%

Source: Infostructure Associates

Object-oriented programming typically increases the time taken during the design phase of a development lifecycle, and decreases the coding time. Added effort to design object classes and define the relationships between them increases design time. Shorter programs and easier code creation from visual designs tend to decrease coding time. Testing may take slightly less time, because object-oriented code is often higher-quality and programs are shorter.

All else being equal, then, object-oriented programming is better than procedural programming for programmer productivity, although worse than higher-level programming. However, in transactional programs, other problems intrude, as described below.

The Object-Relational Mismatch

Much of a programmer's time when creating enterprise applications is spent coding data access, and that much of that time is wasted. Some of that wasted time comes from "rein-

venting the wheel" — writing at a lower level. Some of the wasted time also comes from what Infostructure Associates calls the "object-relational mismatch."

Object-oriented languages produce code that consists of small "object classes" or slightly larger "components", each with the data it uses typically incorporated in the code. Most data, however, lies in relational databases that do not contain the code that accesses the data. As a result, developers must translate in their code between the data-invocation format of an object class and the data structures of a relational database — and the task is rarely easy. Of the 40-50% of coding effort devoted to data access in today's applications, as much as 80% can be devoted to bridging the object-relational divide.

The object-relational mismatch is particularly important when users seek to leverage proprietary information via Web services. In this case, typical Web services development solutions:

1. Hide the details of data access within EJBs.
2. Support standard SQL-based data-access mechanisms, such as ODBC/JDBC and SQL itself.
3. Store the data in the database as objects and provide "object-oriented" data-access mechanisms, such as XQuery.

None of these does as well as higher-level programming that focuses on transactional coding:

1. EJBs often have performance/scalability problems, and are not easily customizable.
2. ODBC/JDBC and SQL are relational, requiring the developer to do the work of object-relational translation.
3. Most data is not stored in object databases, and therefore object-data access is not as well supported by administrators.

Flexibility Problems

When the number of object classes in an application reaches about 1,000 (as often happens when class libraries involving standards are involved), it becomes difficult to impossible for the developer to find the right object class or component for the job. Most of today's development tools do not provide search tools powerful enough to overcome the problem.

Culture and Training

Most old code (up to the mid-1990s) was procedural; most new code is object-oriented. Most of today's databases were around in the mid-1990s. Thus, most programmers who deal with databases are more used to procedural programming, creating cultural and training difficulties. Using object-experienced programmers does not solve the problem, as these are typically less than familiar with transactional coding.

What Genero Is

Four J's Genero includes the Genero "platform" (an architectural layer and components), described on Four J's web site, www.4js.com, and the Genero Studio development toolset (a

high-level programming toolset),. Genero supports rapid application development and upgrade, especially of complex applications.

Genero includes libraries of components that support access to a broad range of data sources, programs (using Genero's Web-services support), and user-interface devices (see Figure 1). Genero also embeds extensive knowledge of "best practices" for database access, communications between tiers of a multi-tier application, business-logic creation, and user-friendly display programming. The Genero high-level programming interface offers a simple syntax for creating user interfaces, data-access code, and business logic. Genero Studio offers Visual Programming Environment (VPE)-style access to Genero's features. As a result, Genero allows users to generate, prototype, and upgrade feature-rich and "rich client" applications swiftly and easily.

Genero is particularly skilled in supporting real-time application tuning and analysis, allowing high ongoing performance and scalability. Genero includes extensive testing tools and automated deployment features. Debugging at the user site and user architectural-information storage make application customization and upgrade much easier — a Genero characteristic of particular usefulness to VARs (Value-Added resellers).

Four J's: Improving Programmer Productivity

The most effective programmer-productivity technologies, as shown in Table 1, are transactional high-level coding, VPEs, and use of high-level, targeted components.

Transactional high-level coding

Transactional "4GLs" use SQL or similar English-language-like descriptive code to access and manipulate data. 4GLs often hide data-access concurrency mechanisms and operate on metadata, rather than on physical data names. As a result, a 4GL can often generate a program directly from the metadata stored in a database's data dictionary (*data-driven design*).

Genero and Genero Studio offer support for high-level metadata, business-logic, and "model"-driven programming. Not only does this support speed development; it also ensures that the data underlying an application can be changed, in structure or format, with little impact on the production application.

VPEs and Components

A Visual Programming Environment (VPE) provides a user-interface-driven, drag-and-drop, point-and-click way for a developer to create a program. Genero provides drag-and-drop user-interface creation plus the ability of users to modify screen fields after deployment.

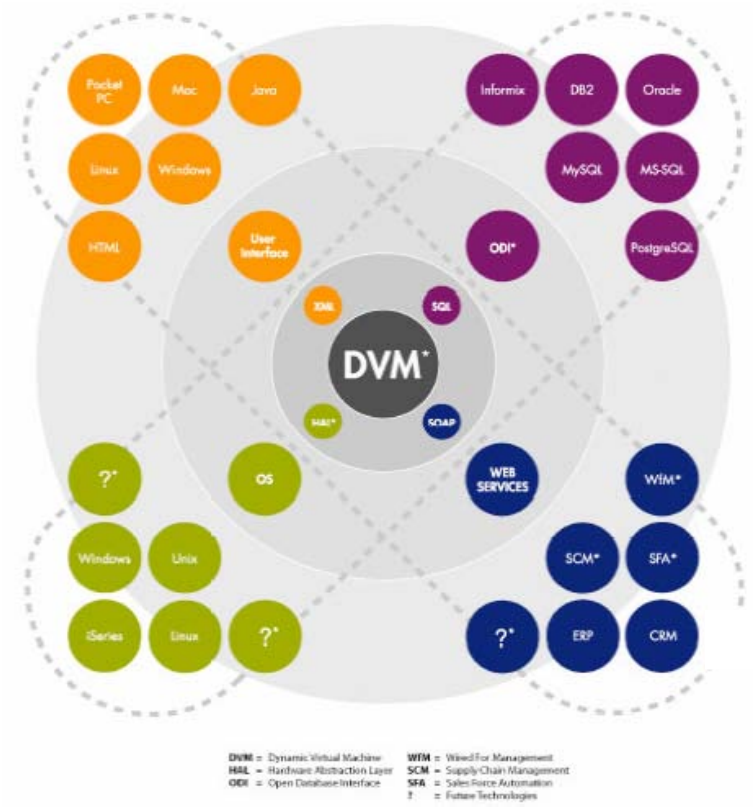
High-level components can provide almost all of the functions needed in an application, with in-house effort creating the "last mile." Genero provides high-level components that interviewees testify are well-suited to their needs and a key factor in speeding development.

Four J's: Improving Flexibility

The amount of money you spend to maintain an application into the indefinite future is usually "baked in" from the start. An inflexible application will require a large amount of support

five or twenty years from now, when no one remembers its structure and hence it is difficult to either fix or upgrade. A flexible application is easy to upgrade incrementally, and therefore is just as useful twenty years ago as it is today, with minimal maintenance costs.

Figure 1: Genero's Architecture



Source: Four J's

Flexibility also has a strong long-term positive relationship to speed-to-value. The easier it is to upgrade a solution, the more rapidly incremental improvements can deliver speed-to-value. Over time, these improvements add up to TCO and ROI benefits equal to or greater than those over the first two years of its life.

Above all, the IT buyer should emphasize an often-overlooked feature of some of today's toolsets — the ability to change existing code to incorporate new features and new technologies, rapidly and easily. Techniques such as refactoring (described below) seem like minor technical tricks today; but are vital to creating the virtuous development cycle of tomorrow.

Thus, development flexibility needs for today's enterprise include:

- *Scalability*, the ability of the application to adapt easily as the number of end users and database size increases.

- *Agility*, the ability to support agile programming and other, similar techniques in order to change development directions, customize, or upgrade rapidly.
- *Architectural flexibility*, the ability to operate on or extend to major popular environments such as Linux, Windows, Unix, Web, Web services, and mainframe/proprietary.

Scalability

Achieving good transactional application scalability typically requires a strong database and a development toolset that embeds a high degree of knowledge about writing transactions (e.g., a 4GL that allows developers to write transactions and business logic at a very high English-language-like level). Therefore, to ensure transactional application scalability, the IT buyer should focus on the right database and toolset, preferably a database optimized for the application's transaction stream and a development toolset that is a 4GL.

Genero's pre-built architectural layer that supports interactions between the end user and a back-end database embeds transactional knowledge that typically improves application performance and scalability. Genero's high-level language generates compact code that inevitably improves not only performance/scalability but also application robustness. Genero's support for "rich clients" allows developers to partition functionality between client and server more effectively for higher performance and scalability.

Agility

Agile programming is an effort fostered primarily by developers that seeks to encourage a different approach to programming, focusing on:

1. Empowering individual programmers and collaboration rather than formal constraints and processes.
2. Interaction with customers rather than following a detailed requirements specification.
3. Fast, incremental changes in response to fine-tuning of requirements rather than long development periods based on a rigid specification.

Refactoring is a way of re-architecting code and data to make it easier to upgrade and understand. Refactoring applied during development of a new application makes upgrading the application in the future much easier.

Genero offers higher-level programming that allows rapid turn-around of incremental changes to an application, and a highly developer-friendly interface that supports rapid prototype generation and allows effective team programming. Genero Studio allows developers to create modules rapidly, so that programmers can respond to end-user requirements or demand changes more quickly. Genero Studio specifies user interfaces and database access at a higher "abstract" level, allowing easier modification. Genero provides — and Genero Studio creates — highly modular, flexible components "pre-refactored" for easy upgrade.

Architectural Flexibility

Today's complex architectures do not lend themselves to easy development. To write a simple Web application (like displaying "hello, world!"), the developer must now write a client,

a server page, and server-side code on top of the application server, plus create a tricky interface to a back-end database. Doing this in Java can require more than 100 pages in a training manual. Some kinds of flexibility therefore come at a price in programmer productivity; the IT buyer should emphasize higher-level programming that allows them to have the cake of programmer productivity and eat flexibility as well.

Genero allows developers to specify user interfaces and database access at a higher "abstract" level. This approach speeds development by avoiding the complexities of the underlying architecture, and enables an application to run automatically across, or extend automatically to, disparate databases and client platforms. Web service support likewise ensures that the application can integrate with other Web services in any Web environment. Genero Studio supports a broad range of standards, clients, servers, and architectures, with easy recompilation to support new front-end devices and on-the-fly end-user modification of display characteristics. Thus, users have a wide choice of application characteristics and interfaces.

The Importance of Software Evolution

The effective IT strategist looks at the corporate in-house and packaged application portfolio — indeed, the whole corporate software portfolio — as a key corporate asset. Thus, a software portfolio asset must be maintained; should be constantly improved by judicious investment; can be judged and fine-tuned for cost-effectiveness.

A frequent error of IT strategists has been to look at maintenance, investment, and cost-tuning separately — to budget for new functionality as a "make or buy" decision without considering the possibility of upgrading existing software; to treat maintenance as a matter of fixing bugs rather than investing in upgrading the application to support new uses; to seek to cut maintenance costs (e.g., by outsourcing) without considering upgrading the application to more robust, less costly technologies. The result has been a "vicious circle" of increasing costs: IT does not invest in upgrading an application; as this application falls further behind today's technology, costs of maintenance increase and knowledge of the application decreases; lack of knowledge of the application makes upgrade harder; the difficulty of upgrade makes investment less appealing.

In order to avoid this "vicious circle," IT should plan for *software evolution*; i.e., upgrade software where appropriate rather than simply maintaining it. More specifically, IT should:

- Do a periodic "software audit" that lists existing applications and software, in-house and packaged, and note which software is not yet Web-enabled and/or Web-servicized.
- Use the audit to divide the software into three categories: (1) software to be replaced by new in-house or packaged software; (2) software to be improved; and (3) software to leave untouched for a while.
- Aim software improvements especially at bringing the software into synchronization with the latest technology — and particularly by Web-enabling and Web-servicizing it.

Software "improvement" or upgrade may include:

- "Upgrade in place." In this case, the developer leaves the application on its present platform (e.g., the mainframe) but either "encapsulates" it with a Web or Web-service-provider front end or changes its code in order to incorporate new technology or capabilities.
- Migration or porting. In this case, the application may be moved to a new platform (e.g., mainframe to Unix or Windows) and integrated with the new technology on the platform. Although this would seem to involve little change to the existing application, many applications are highly dependent on the operating system or infrastructure that they invoke, and migration may involve extensive changes. A variant of this approach moves the application code to a new platform but leaves the application's data store on the old platform.

The Benefits of Software Evolution

The key benefit of a software evolution strategy is that it turns the "vicious circle" of ever-increasing per-application maintenance costs into a "virtuous circle" of decreasing maintenance costs as new technologies improve application robustness. Other benefits include:

- *Increased reuse of existing code*, resulting in increased programmer productivity. Upgrading legacy applications increases the knowledge of these and allows "componentization" of existing applications, making them more suitable for reuse.
- *Decreased administrative and training costs*. In many cases, upgrading legacy applications allows users to move to fewer systems and platforms, decreasing the expertise needed for application administration.
- *Increased strategic flexibility*. IT now not only can adapt more quickly to end users' changing needs by tweaking existing applications, but can create more new functionality by combining existing functionality, e.g., by creating composite applications that combine business processes by invoking the applications that carry the processes out.
- *More information about the software portfolio asset*. Often, the application upgrade processes unlocks previously undiscovered information about an application (e.g., the existence of business rules) — information that can be used for better application auditing and monitoring, to improve the in-house development process, and to drive IT strategies.
- *Another development choice besides "make or buy"*. In many cases, upgrade may be preferable to either.
- *Better leveraging of information locked in proprietary applications*. Upgrading typically allows wider and more effective end-user access to key business information.
- More rapid development of new applications, by basing them on a development platform that supports a long-term, evolutionary software strategy.

Implementing the Software-Evolution Strategy

Many large enterprises, aided by services from suppliers such as IBM and Unisys, have gained extensive experience in software evolution or "legacy modernization." These experiences indicate that the choice of the right toolset is vital to upgrade success. The toolset and its "platform" (set of components and infrastructure software) should have:

- Support for Web-enablement and Web-servicization.
- The ability to operate at a high level via abstraction (i.e., higher than a 3GL like C# or Java) so that migration from one environment to another, or upgrade of existing code, is a matter of changing a few high-level parameters (such as the platform involved) rather than extensive changes to not-always-well-documented code. This is sometimes referred to as "abstracting," omnicompetence (operating at a high metadata level), or operating at the model/design level.
- The ability to store information about the application, such as its platform, metadata, and process abstraction, for ongoing use in further upgrades.

Four J's Genero and Genero Studio meet these criteria, as shown above:

- Interviewees testify that it is "straightforward" to implement Web and Web-service capabilities using Genero and Genero Studio, especially given Genero's support for host-based and client-server as well as Web and Web-service applications.
- Genero Studio plus Genero's architecture and components allow the programmer to operate at a high coding and metadata level.
- Genero Studio stores information about the application that makes upgrade in place or migration easier.

Note that Genero and Genero Studio are especially appropriate as a target platform for an existing-application rewrite aimed at giving users the flexibility to invoke new technologies rather than being locked into a legacy environment, as well as to modernize an Informix-4GL-based application.

Genero TCO/ROI

This section summarizes the results of several Infostructure Associates studies comparing the total cost of ownership (TCO) and return on investment (ROI) of higher-level development solutions, such as Four J's Genero/Genero Studio, to so-called third-generation language development tools, such as those offered for the C# programming language by Microsoft in Visual Studio .NET 2003 with its .NET Framework platform or for Java by Sun Microsystems, Oracle, and IBM. It also assesses TCO and ROI for ISVs (Independent Software Vendors) selling business applications. For further details, see the Appendix and the Infostructure Associates report, "Web Services Development Solutions Report: Speed to Value", September, 2004, www.valleyviewventures.com.

The reader should note that this qualitative research involves ongoing in-depth interviews of enterprises with real-world comparative experience of two or more of the development solutions involved. Often, the interviewees are ISVs with experience of the development solutions involving several projects over several years.

The findings of the studies are:

- *Higher-level development solutions such as Genero demonstrate a clear advantage in both TCO and ROI over 3GL tools in a wide variety of situations.* Infostructure Asso-

ciates research suggests *compared to Java/Oracle use, Genero yields TCO reduction of almost 300%, on average, and ROI improvements of almost 100%.*

- *These advantages are especially marked in larger-scale and mass-deployment architectures*. In these architectures, administrative costs dominate over time, and the higher-level toolset's ability to deliver more robust solutions with lower administrative costs makes it especially attractive. This superiority is due both to the software quality of higher-level-toolset-developed applications and to sharply reduced database administrative costs.
- *User need for speed to value has made the effectiveness of a development tool more important than at any other time in the last eight years*. Our research shows that speed to market is now heading the list of key advantages of a variety of new technologies, ahead of cost savings for the first time in recent memory. Users testify that the effectiveness of a development solution is the major determinant of speed to value.
- *Increasingly, users are turning to implementing better access to competitive-advantage proprietary information, as opposed to the traditional method of creating new competitive-advantage functionality by making or buying a new application, to achieve long-term competitive advantage.* As a result, the TCO and ROI advantages of data-savvy development solutions versus data-indifferent 3GLs are likely to increase over the next few years.

Comparison Research Results

The following tables list TCO and ROI figures for 3GL and higher-level development solutions creating new mission-critical applications. All calculations are rounded to the nearest dollar.

Table 2 shows estimated three-year TCO for Oracle's Java development solution and the Four J's Genero solution, in the 25-, 50-, and 50x20-user cases. Table 3 shows the estimated ROI for these solutions.

All costs are listed as a total cost for the three-year period, whether they are actually one-time costs (e.g., development costs and software license costs) or costs that accumulate over three years (e.g., internal maintenance costs). Likewise, benefits are listed as a total for the three-year period. Maintenance costs are assumed to stay level over time. Infostructure Associates assumes that all development solutions allow the application to be completed and deployed within two years from project start.

Table 2: Microsoft and Four J's Development Solution Three-Year TCO

	25 Clients	50 Clients	50x20
Database License (Framework)			
Genero solution – assume Microsoft SQL Server	$9,998	$19,996	$499,990
Oracle Database Standard Edition 10g	$15,000	$30,000	$750,000
Application Server License (Framework)			

Genero solution – assume Microsoft Application Server	$0	$0	$0
Oracle Internet Application Server Standard Edition	$10,000	$20,000	$500,000
Development Environment (ten copies)			
Genero solution	$50,000	$50,000	$50,000
Oracle Internet Developer Suite (Java)	$50,000	$50,000	$50,000
Development Cost (10 Programmers)			
Genero solution	$700,000	$700,000	$700,000
Oracle Internet Developer Suite (Java)	$1,035,000	$1,035,000	$1,035,000
Two Application Upgrades (new releases)			
Genero solution	$140,000	$140,000	$140,000
Oracle Internet Developer Suite (Java)	$207,000	$207,000	$207,000
Deployment Cost			
Genero solution	$6,500	$12,000	$17,500
Oracle — with $4,000 application server deployment	$47,499	$98,999	$193,998
DBA Cost			
Genero solution – assume Microsoft SQL Server 2000	$17,000	$33,000	$240,000
Oracle Database Standard Edition 10g	$150,000	$150,000	$7,500,000
Application Server/Web Server Administration Cost			
Genero solution – assume Microsoft Application Server	$15,000	$15,000	$15,000
Oracle Internet Application Server	$50,000	$50,000	$50,000
Development Toolset Training			
Genero solution	$16,000	$16,000	$16,000
Oracle Internet Developer Suite (Java)	$32,000	$32,000	$32,000
Platform Training			
Genero solution -- Microsoft training	$7,280	$7,280	$7,280
Oracle Application Development Framework	$5,000	$10,581	$137,553
Two Platform Upgrades over Three Years			
Genero solution –assumed like Microsoft	$5,549	$5,549	$5,549
Oracle	$1,500	$3,000	$75,000
Support/Maintenance			
Genero solution –assume like Microsoft	$6,000	$12,000	$300,000
Oracle	$700	$1,400	$35,000
Total Cost of Development Solution Ownership			
Genero Solution	**$973,327**	**$1,010,825**	**$1,991,319**
Oracle Internet Developer Suite/App. Framework	**$1,603,699**	**$1,687,980**	**$10,853,551**

Source: Infostructure Associates

Table 3: Microsoft and Four J's Development Solution Three-Year ROI

	25 Clients	50 Clients	50x20
TCO			
Genero solution	-$973,327	-$1,010,825	-$1, 991,319
Oracle Internet Developer Suite/App. Framework	-$1,603,699	-$1,687,980	-$10,853,551
Opportunity Cost Savings			
Genero solution	$76,000	$75,900	$75,900
Oracle Internet Developer Suite/App. Framework	-$110,215	-$124,250	-$2,579,773
Business Benefits			
Genero solution	$7,333,000	$14,666,000	$355,333,000
Oracle Developer Suite 10g	$4,700,000	$9,400,000	$235,000,000
Return on Development Solution Investment			
Genero Solution	**$6,435,673**	**$13,731,075**	**$353,417,581**
Oracle Internet Developer Suite/App. Framework	**$2,986,086**	**$7,587,770**	**$221,566,676**

Source: Infostructure Associates

Additional Infostructure Associates Key Research Findings

IT buyers should note the following additional findings from this study:

- *A good development toolset is an extraordinary bargain*. For an additional $20,000, or so, IT buyers can achieve $300,000-$600,000 in three-year cost savings, and $3 million-$35 million in additional return on investment, for the mission-critical applications studied here. Note that a small application with only 25 end users can yield $300,000 in cost savings and $3 million in additional ROI, while a large "mass-deployment" application with 1000 end users can yield $600,000 in cost savings and $35 million in ROI. Note also that this cost savings/ROI is *for each application created*.
- *Choosing the right embedded infrastructure/framework for mission-critical application development and runtimes can yield major additional cost savings and benefits*. Approximately 10% of the value-add from the development solution derives from choosing a framework that fits well with the toolset and provides high-level components appropriate to the needs of a vertical industry or set of business functions.

Conclusions

The advantages of high-level programming, not only to IT spending but also to the corporate bottom line, via speed-to-value competitive advantage, cost-cutting, and business continuity for risk mitigation, are becoming ever stronger and clearer. However, major popular 3GLs such as the Java and Microsoft toolsets lag in delivering high-level programming to all but a small minority of present-day IT shops and ISVs. As a result, IT buyers should look to real-world-proven third-party suppliers — such as Four J's, with its Genero/Genero Studio development solution.

Infostructure Associates research shows that Genero delivers exceptional high-level programming and high-level component/platform support, strong support for upgrade and customization flexibility, and comprehensive support for a software evolution strategy. As a result, our qualitative research suggests that Genero can provide a 20-25% advantage in TCO and a 30-35% advantage in ROI over 3GLs, along with application scalability into the terabyte-database range and increases in application availability and quality. Its Web services support has proven itself in large-scale real-world implementations, and ensures semi-transparent incorporation of the Genero solution and Genero-developed applications in enterprise architectures.

Speed to value cannot be achieved simply by cutting developer costs via offshore outsourcing, or by "throwing more bodies at the problem" (else, as Frederick Brooks noted long ago, it would take 9 women one month to produce a baby!). Without a high-level development solution, speed to value is difficult or impossible in most cases; with it, more and more enterprises are achieving competitive advantage in the real world. Therefore, IT buyers and ISVs should place solutions such as Four J's Genero/Genero Studio at the top of their shopping lists.

Appendix

Infostructure Associates' Development Solution Model

Infostructure Associates research shows that creators of mission-critical applications now typically employ not only development tools but also a "framework" or "platform" of software infrastructure exposed to the developer as components. The developer invokes these components to create key parts of the application, and when the application is deployed it "runs on" (i.e., uses during runtime) the software infrastructure itself. A typical framework includes:

- Business-logic components.
- A database.
- A second-tier Web server.
- A second-tier application server, typically with load balancing and server-side development support.
- Database, application server, and application administration tools.

Because only the database and application server costs are significant, the Infostructure Associates model includes these as framework costs.

Frameworks may also include functions such as data integration tools, security utilities, enterprise portal functions, or Web services support (we do not include these in our computations because they are not yet significant contributors to TCO or ROI).

Factoring In Scalability and Mass Deployment

Many applications are now deployed in a "mass-deployment" architecture:

- Multiple distributed copies of the application and database, each invoked locally by fewer desktop and mobile client devices (the mass-deployment model). The application typically carries out data sharing between copies by a small amount of synchronization (via replication) with a single central database.

Infostructure Associates' mass-deployment model includes software suppliers and service providers deploying an application to multiple customers, each with a workgroup (e.g., an insurance application deployed to individual branch offices), as well as larger organizations deploying an application to remote offices.

Computing TCO and ROI

TCO includes cost of software, development, deployment, upgrade, support, and administration of the application. ROI compares a 3GL toolset to higher-level development solutions as well as to "doing nothing", and equals opportunity cost savings, plus the dollar value of additional business benefits, minus TCO.

Opportunity cost savings assumes that TCO cost savings are reinvested in projects with a 30% ROI over three years. Note that where business benefits are high and can be reinvested, this approach underestimates the effect of opportunity cost savings. Business benefits derive primarily from "speed to market" — that is, software quality, flexibility, and scalability have a much smaller ability to increase sales and margins than programmer productivity for a 3-4 year application lifecycle.

IT buyers can use TCO figures to get some sense of likely cost savings, as well as the relative cost-of-ownership of various solutions. By contrast, business benefits of an application may vary widely depending on the type of application and type of business. Therefore, IT buyers can use ROI figures to compare between development solutions, but should not assume that ROI figures will approximate users' actual results.

IT organizations need to be aware of the real costs and benefits associated with creation and deployment of an application — including opportunity costs. While cost of ownership today can be an important factor in the evaluation process and the purchasing decision, ROI is typically less used and less trusted. However, because users now care about speed to market, it is important for readers to consider development solution ROI as well — speed to market of an unimportant application may lead to negative ROI.

Project Methodology

Infostructure Associates' qualitative research findings are based on interviews with enterprises using "embedded infrastructure" (i.e., platforms with databases) to create new applications, as well as ISVs; extensive research into the programmer-productivity characteristics of various types of development tools; and case studies of recent users of both a 3GL and a higher-level development solution.

What Was Included in Each System

In calculating the TCO of a development solution, Infostructure Associates included the following:

- *Development solution license*
- *Database and application server (or framework) license*
- *Development and upgrade costs*
- *Deployment costs*
- *DBA and application-server administration costs*
- *Development solution and framework training costs*
- *Two platform upgrades*
- *Two application upgrades*
- *Support/license maintenance costs*

Infostructure Associates
113 Bristol Road
Wellesley, MA 02481
USA
Telephone: 508 265 1558
Fax: 781 862 6236
www.valleyviewventures.com

Infostructure Associates is an affiliate of Valley View Ventures that aims to provide thought leadership and sound advice to both vendors and users of information technology.

This document is the result of sponsored research performed by Infostructure Associates. Infostructure Associates believes that its findings are objective and represent the best analysis available at the time of publication.

About the analyst: Wayne Kernochan is a long-established expert and thought leader in the database, development solutions, and Enterprise Information Integration (EII) areas. He has been a strong advocate of strategic information management and agile programming. In 2002, 2003, and 2004-5 he has issued yearly database, development, and EII reports as well as in-depth qualitative research in these areas, including TCO/ROI and case studies.

During an eleven-year tenure at Aberdeen Group, Kernochan was the author of their unique approach to TCO studies and held the positions of senior director, managing vice president, and senior vice president. Kernochan is frequently quoted in publications such as Information Week and CIO Magazine.

Email the analyst:
wayne.kernochan@valleyviewventures.com