

component • development strategies

The Monthly Newsletter from Cutter Information Corp. on Managing and Developing Component-Based Systems

Executive Summary March 2001

In this first issue under my editorship, I'll review the evolving potential of distributed component technology to meet today's increasing business challenges. I'll focus on some recent developments in component technologies, including the software component market, Web services, integration technologies (such as application servers), and component standards. Unfortunately, most methodologies fall well short of the bar that has been raised by these advances; the accompanying transition in organization, architecture, and software process has been slow to emerge. I'll conclude by illustrating some of the shifts needed. Next month, I'll look at light methodologies as a response to this problem.

As an enthusiastic reader of CDS for many years, I'm excited to be taking over the editorship from Paul Harmon. I have long appreciated Paul's wealth of knowledge and perceptive insights and wish him well. I am conscious of the high standards of editorship that CDS readers have enjoyed and begin my first issue of a new era with a sense of humility.

Contents

The State of the Practice	1
A Brief Historical Picture	1
Component Concepts	5
Enabling Technologies	7
Component-Based Strategies	14
Summary	16



Paul Allen, Editor
E-mail: pallen@cutter.com

THE STATE OF THE PRACTICE

This issue of *CDS* provides an overview of the business-IT landscape and is designed to set the scene for the months ahead, giving readers a sense of the challenges and opportunities of component technology. I'll start by painting a quick historical picture to put things in context. This is very much a picture of increased pressure to quickly deliver effective software solutions that are capable of withstanding the test of time and the challenge of change. The landscape is one of increasing diversity of choice in terms of components, frameworks, Web services, packaged software, and a range of infrastructure technologies that vie for the attention of the beleaguered chief technology officer. Much of this issue is devoted to surveying and commenting on these technologies, particularly fast-emerging Internet-related topics such as Web services and the component marketplace.

Although today's software projects share much in common with their ancestors, such as tight deadlines and difficulties understanding requirements, there are some important differences. A key difference is that projects today are seldom pure development projects; rather, investments in existing systems must be realized, and you must also take advantage of an increasing array of commodity software available on the open market. At the same time, there is pressure to partner across the supply chain, which takes us away from development and into an era of collaboration. Unfortunately, these factors are dawning very slowly in our industry — dated methodologies still prevail, often with disastrous consequences.

A Brief Historical Picture

E-business was covered thoroughly in two recent editions of *CDS* (see October 2000 and December 2000). Therefore, in this issue, I just want to provide an idea of the e-business opportunities and challenges that serve as push factors behind distributed component computing.

During the past four decades, IT has gone through topology shifts that initially supported, then mirrored, and are now set to converge with the business organization. I've summarized these shifts in Figure 1. In the 1960s and 1970s, IT was a back-office operational resource supporting departmental business silos. The change to client-server computing in the 1980s and early 1990s saw IT move into the era in which IT was split into database back-end

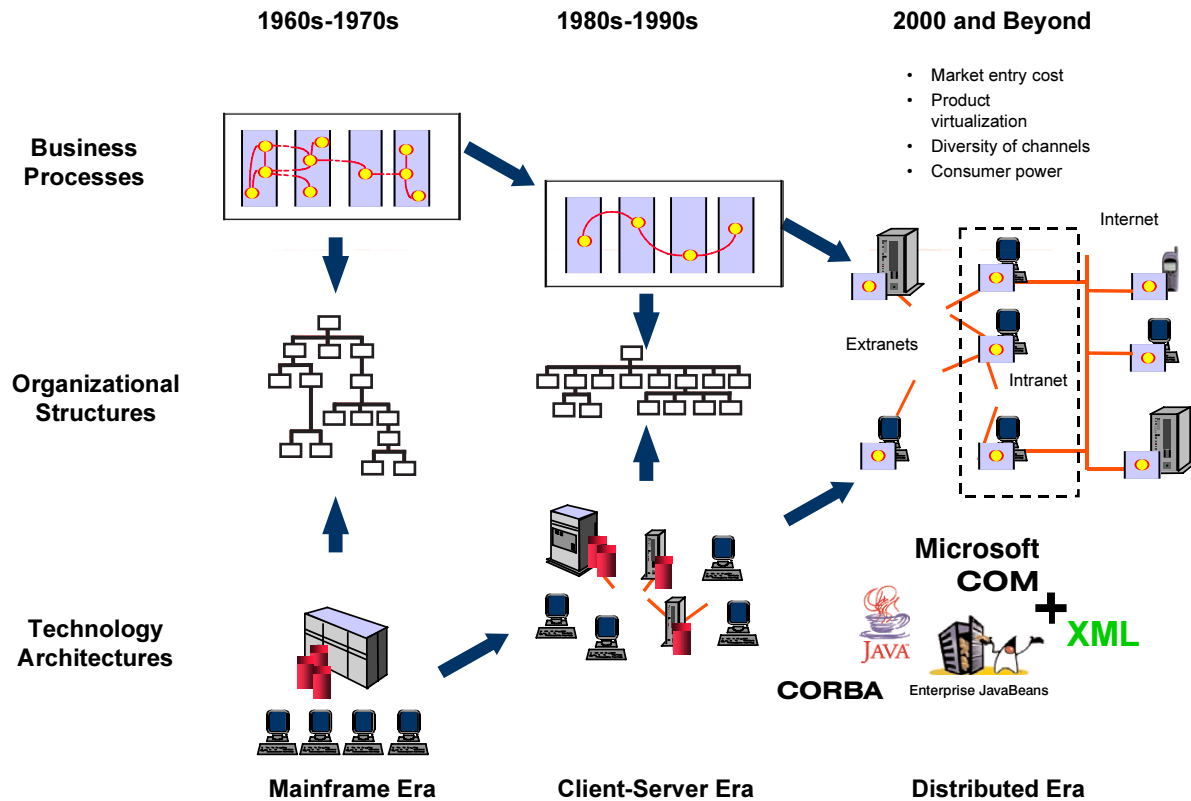


Figure 1 — The convergence of business and IT.

groups servicing front-end programming groups. The front-end responsibilities started to migrate to the business units in the latter half of the 1990s, driven by an increasingly competitive marketplace and enabled by decreasing unit costs of basic technologies (e.g., processors, storage, and bandwidth). In parallel, business process reengineering related efforts to leverage the technology saw businesses become more process-oriented in an effort to remove the “white space” on the organizational chart. The 1990s saw the upsurge of distributed computing and the explosion of the Internet. From a business viewpoint, the concept of a monolithic enterprise owning all the products, services, and channels required to address a customer’s needs is rapidly disappearing.

Growing consumer demand is reflected in soaring organizational demand for software that provides business efficiency, competitiveness, and innovation. As customers grow more selective and powerful, virtual enterprises and integrated value chains replace the traditional organization. Enterprise boundaries that were once very much fixed (as depicted by the two solid-lined rectangles in Figure 1) are increasingly fluid (as depicted by the dashed lined rectangle) as companies merge and partner to exploit new opportunities and strive for flexibility. It’s no longer organizations that compete, but supply chains of collaborating organizations. And the same organization may participate in different supply chains in different spheres of competition.

Editorial Office: Paul Allen, Tel: +44 1737 813911; Fax: +44 1737 814279; E-mail: pallen@cutter.com.

Circulation Office: *Component Development Strategies*® is published 12 times a year by Cutter Information Corp., 37 Broadway, Suite 1, Arlington, MA 02474-5552. Tel: +1 781 641 9876 or, within North America, +1 800 492 1650; Fax: +1 781 648 1950 or, within North America, +1 800 888 1816; E-mail: service@cutter.com; Web site: www.cutter.com/consortium/.

Editor: Paul Allen. Publisher: Karen Fine Coburn. Group Publisher: Kara Lovering, +1 781 641 5126, E-mail: klovering@cutter.com.

Production Editor: Pamela Shalit, Tel: +1 781 641 5116. Subscriptions: \$497 per year; \$567 outside North America. ©2001 by Cutter Information Corp. ISSN 1552-7391. All rights reserved. Unauthorized reproduction in any form, including photocopying, faxing, and image scanning, is against the law. Reprints, bulk purchases, past issues, and multiple subscription and site license rates are available on request.

Organizational Factors

Changes in the IT workplace have paralleled those occurring in the business and technology worlds. In the 1960s and 1970s, fixed hierarchical management structures reinforced an environment of ordered stability that could border on the bureaucratic. Sometime in the mid-1980s, this world started to change irrevocably. As businesses started to downsize and streamline their operations, so did IT. By the mid-1990s, IT was growing increasingly diverse, both organizationally and geographically. At the same time, businesspeople started to take a much more central role in software development, and IT people needed to apply a much greater level of business knowledge.

Increasingly, many of the traditional development tasks of IT departments are being outsourced to specialist application service providers (ASP). This change puts the onus on internal IT development staff to focus on value-added activities that confer competitive advantage. Similarly, it creates an environment in which IT managers must switch toward overseeing the broad strategy; they must look away from projects and toward programs. Equally, e-business puts the onus on IT managers to be proactive in business process improvement. Technospeak is unlikely to receive much sympathy in the boardroom. The language of IT increasingly needs to be the language of business.

The Illusion of Business-IT Alignment

Of course, the picture I have painted is an idealization. Change, once viewed as a short period of transition between two (much longer) periods of relative stability, is now a continuous process. Unlike the closed systems of the mainframe, client-server, and early distributed eras, software is (increasingly) never “complete.” And while that’s a challenge in itself, the reality is that most organizations have a mix of technologies and business practices that reflect different eras. The operational and tactical systems of earlier days — the legacy systems — still have a vital part to play. They must be made to work hand in glove with the new systems interconnected to those of suppliers and partners. Not only that, they must also be efficiently maintained. “New” legacy systems in technologies such as Visual Basic and C++ add to the groundswell.

This picture is also an idealization in that it assumes alignment of business and IT. There’s an important distinction to make here. On the one hand, the technology used nowadays *is* increasingly the business solution — in that sense, business and IT are converging. Bank ATMs were one of the first examples of this; today it is possible in some parts of the world to buy a Coke from a vending machine using a cell phone. On the other hand, poor customer experience is still common. I’m reminded of my own frustrations trying to order items online recently; in two separate instances, ordering supermarket goods and buying rail tickets, I entered all my information online only to have to submit a fax days later when my online order encountered problems. The root of such problems is often a failure to align IT solutions with business needs. The result is e-business systems that may converge with business but fail to align with it, as illustrated in Figure 2. Business processes may appear to map cleanly to technology solutions, but they are too often fragmented and lack integration. For example, many e-business systems use Web-server databases to help manage content and provide limited customer-related information. Nevertheless, these solutions are essentially standalone. The consequences of manual rekeying of information at any point in an e-business process are not difficult to visualize in terms of incorrect information. Equally important, if the same manual steps are required to process orders as they are for an existing channel, such as mail order sales, then a Web front end is not a way to improve efficiency and reduce operational cost. On the contrary, the Web front end becomes an additional overhead.

The Puzzle of Business-IT Misalignment

The shakeout of early dot-coms is well documented. Although “new economy” forces such as diverse channels and consumer power are a challenge in their own right, the traditional “old economy” forces such as cost reduction are still there, and they have to be balanced with the new. For example, there were a number of retail dot-coms that were forced out of business last year as a result of industrial action at one of the global goods delivery firms.

The upshot is that it is only now that most companies are gearing up to build really large e-business systems that integrate with partners and tackle the complexities

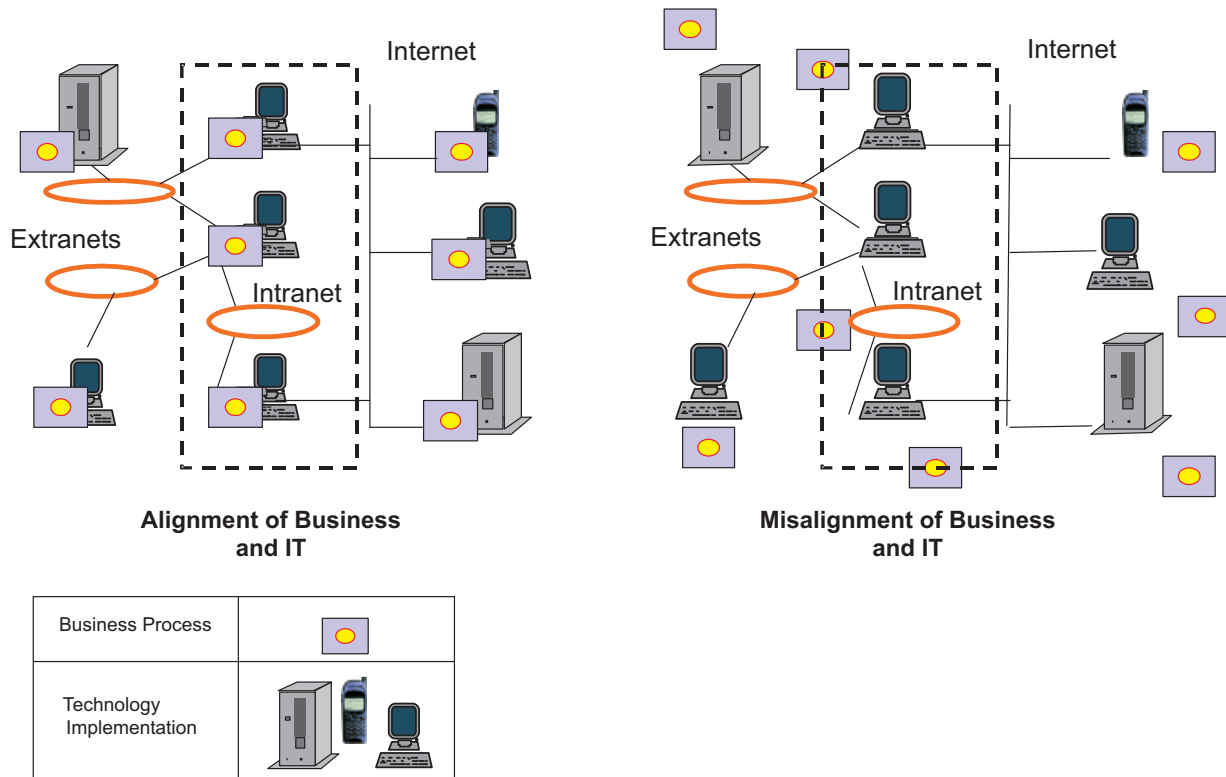


Figure 2 — Alignment versus misalignment of business and IT.

of full business process integration. Major efforts are needed to integrate legacy systems that were never designed to work in Internet time. Adding to the complexity of the situation is the fact that company applications must be connected to entirely different applications created by suppliers and partners across a mix of technologies that have accumulated over the past three decades.

All in all, there is a lot of pressure on the architects, analysts, and developers of today to produce critical e-business solutions before some competitor does. In this case, one of two things can happen. One scenario is that planning, analysis, and design are often simply abandoned in the name of the deadline; instead, super programmers get the project off the hook yet again! Such short-term thinking results in solutions that may look good at first but soon become part of the maintenance backlog as the super programmers leave, taking their hard-won knowledge with them. As the organization goes through more changes, business and IT grow increasingly misaligned.

The second scenario is that techniques are used from one of the previous eras geared to producing point solutions for internal departmental needs. Worse still, rigid and inappropriate processes and architectures are applied. The result is often severely limited applications that again lock the organization's hands (albeit in a year's time) as they fall out of alignment with business needs. In fact, this is worse than the first scenario because of the excessive time taken to build and maintain the accompanying models and architectures.

The Role of Components

Another aspect of the historical picture is that through each era, previously low-level programming activity is abstracted to higher levels until it is productized and made part of the software infrastructure. This happened first with assembly languages, then operating systems, and later with application packages. While components are part of this same trend, they take things significantly further — from a world where operating systems are the dominant factor to a world where standards are the dominant factor. They hold the promise of effective

management of complexity, reuse of proven functionality, and software that adapts as the business adapts to provide business insulation from technology change. Here, we are told, is an underlying fabric for the brave new world of e-business in the form of flexible services that can be reused in different combinations in the manner of loosely coupled and independently managed units.

This is a great vision; it is a vision I see taking shape in my work with different organizations as they increasingly turn to components as a way to encapsulate existing functionality, acquire third-party solutions, and build new services to support emerging business processes. And it's a vision that is backed up by some impressive advances in enabling technologies that I'll focus on shortly. But how far are we down that road? We must first take a short detour to look at some basic concepts.

Component Concepts

There is still much debate about the meaning of the term component, and, although I don't want to get sidetracked, I think it's important to define some basic terminology before we go much further.

The Anatomy of Components

A good place to start is with the following definition from Clemens Szperski's *Component Software: Beyond Object-Oriented Programming* (Addison-Wesley, 1998), which is pretty widely accepted: "A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Context dependencies are specified by stating the required *interfaces* and the acceptable execution platform(s). A component is subject to composition by third parties. For the purposes of independent deployment, a component needs to be a binary unit."

Taking this definition further, there are some additional aspects I'd like to discuss. For example, interfaces are well-defined sets of software services that provide access points to clients or users of components; a user may be human but could also be another component that is dependent on the interface. Interfaces do not depend on the technology used to implement the component. They are "technically neutral." Components are software units that offer sets of one

or more interfaces and whose functionality can only be accessed through those interfaces. A component can be implemented using different technologies to support the same set of interfaces. Components must have software plug points that fit into sockets provided by a component execution environment (CEE). A CEE is required to provide runtime technical infrastructure services and to hide low-level technology issues from the business solution developer. The CEE offers its infrastructure services through sockets — software written to well-defined and well-known standards into which components can plug. Figure 3 illustrates these concepts. The CEE complies with a component specification and a set of standards, including a component meta model.

There are client-oriented component meta models, and there are server-oriented component meta models. A number of competing component infrastructures exist based principally on one of three technologies: Microsoft's COM, Sun's JavaBeans, and the Object Management Group's (OMG) CORBA. Each technology supports a client model and a server model (see Table 1).

Another important feature of components is the physical packaging of components into executables. An executable may contain several components. Conversely, a component can be contained in several different executables. It is executables that are deployed on machines. This provides great flexibility in that components can be packaged in different ways according to the capabilities of the runtime environment and the constraints upon it. Components may be used individually or bundled together to create new solutions and rapidly develop business advantage. Unfortunately, a number of approaches either do not recognize or blur the above distinctions; this is a subject I'll return to in a future issue.

Business Components

Business components are distinct from GUI components such as drag-and-drop buttons, boxes, and images. They are also distinct from utility components used to provide system and infrastructure services, such as database connectivity and event service components. In contrast, business components encapsulate business characteristics such as business logic, business process,

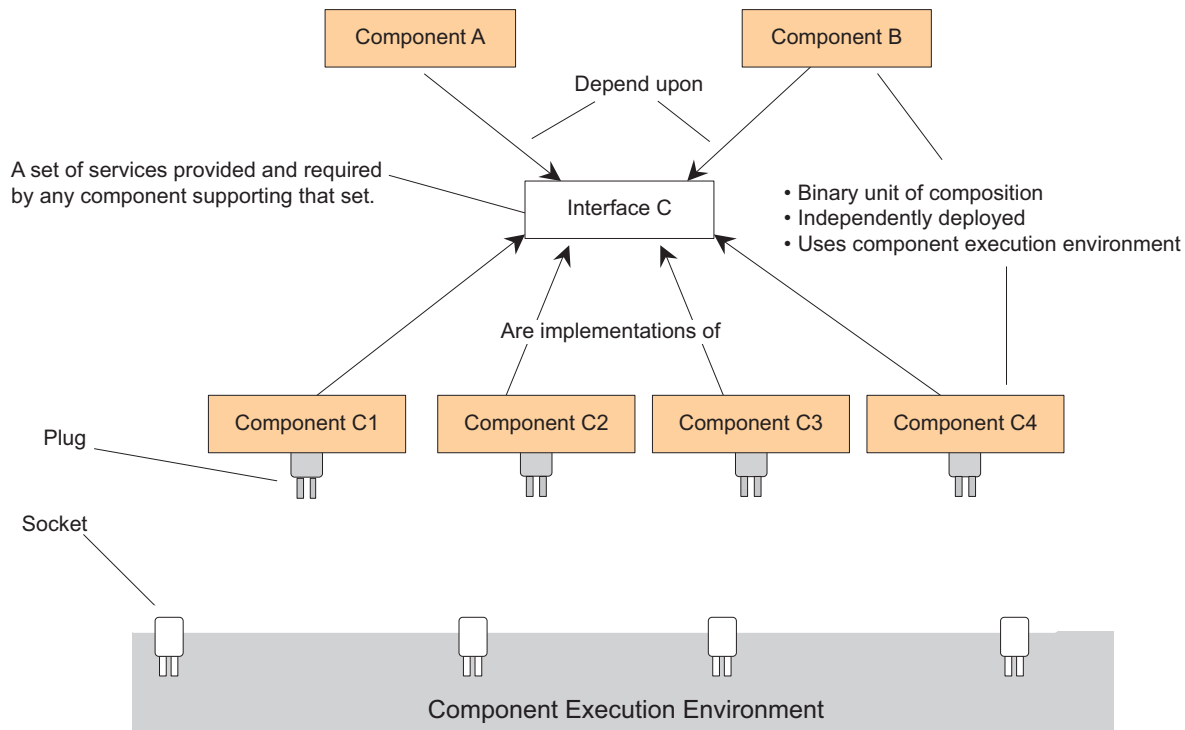


Figure 3 — Anatomy of components.

business rules, and information. Thus far, most component reuse has centered on reuse of technical components, especially GUI components. The power of business components, which provides the capability to make real inroads into better productivity, resides in the business knowledge that goes into developing those components. This knowledge is embodied in the component models that are used to architect and specify the business components.

Components Versus Objects

To a large degree, components have grown naturally out of developments in object technology. Components are like objects in that implementation is encapsulated. However, a component is unlike an object in that encapsulation is guaranteed. Implementation dependencies are often exposed in OO programming languages. This is not true of components. Someone once said that a component was rather like an object in a dinner jacket, which seemed a cool way of putting it. More significantly, the implementation language can be anything we choose. It can be an OO programming language, but it

can just as easily be a previous generation structured language like COBOL or PL1.

Components Versus Frameworks

It's also important to be clear about the distinction between components and frameworks. If a component is a black box that can only be addressed through its interface, then a framework is a white box — we can tinker inside it until it is just right. In the context of a component (as opposed to an object), this means we can extend the interface, not the internals of the component, which remain hidden. The framework may well be based on patterns that have been used to guide its creation.

Components Versus Patterns

A pattern is reusable analysis or design knowledge that describes a problem and its recommended solution. A good pattern is concise: small but packing a heavy punch. This helps an experienced practitioner recognize a pattern, including its core concepts, so that it can be adapted to the specific requirement. Although design patterns have received the most attention, the

Table 1 — Component Meta Models

	Microsoft	Sun	OMG
Client-oriented	COM: OLE, OCX, ActiveX	JavaBeans	CORBA Object
Server-oriented	Microsoft Transaction Server/COM+	Enterprise JavaBeans/ Java 2 Enterprise Edition	CORBA Component Model

concept applies throughout the software process. For example, the same idea applies at the level of software architecture and requirements knowledge. Architecture and analysis patterns have been around for a while now. Similarly, patterns that provide process guidance have gained in popularity.

Space constraints limit me to a very brief account of these key concepts in this issue. While I'll cover these in more detail in future issues of *CDS*, the point I want to emphasize right now is that an effective component-oriented approach should be broad enough to exploit them all. Success will also depend on effective modeling of components, in this broad sense. This in turn depends on good visual modeling tools with an integrated repository for underlying "once only" definitions of model items.

Enabling Technologies

Projects no longer start with a clean slate nowadays. How many times are we charged with integrating this or that software package, reusing or improving systems produced by other projects, supporting programs of collaboration with partners, or linking to suppliers' systems in mid-process?

The fact is, there is a rapidly growing set of resources that are an increasing feature of the software landscape today. For example, there are what I'll loosely call integration technologies, from enterprise application integration (EAI) tools to application servers. These provide facilities to unlock the potential of legacy assets such as software packages, databases, and legacy systems. There is the software marketplace itself, reflecting a host of possibilities from traditional enterprise resource planning (ERP) vendors to open source, from ASPs that want to run your IT shop to a variety of online component brokerages. And there is the emerging area of Web services that promises easy

access of proven business functionality over the Internet. Let's take a look at how these elements are both competing and partnering to become part of your solution.

Integration Technologies

What I'll broadly refer to as "server-based architectures" have grown largely out of the huge increases in scale (for example, in terms of transaction rates) and in the variety of channels to market that have emerged over recent years. Underlying these increases is enormous pressure to integrate a wide and rapidly growing variety of back-end servers and application systems.

Unfortunately, the term "server" is used in a variety of ways, even by the same vendor — e.g., Windows 2000 Server, Microsoft Transaction Server (MTS), Microsoft Message Server. There are HTML servers, Enterprise JavaBeans (EJB) application servers, database servers, and e-commerce servers. IBM is increasingly referring to its mainframe systems as servers. I find it helps to approach this topic by logical layers of functionality. This provides at least some kind of benchmark against which to make sense of an evolving situation, unravel vendor claims, and compare different products. There are four broad layers:

1. **Client environment.** Typically, a Web browser supports HTTP/CGI/Applet/ActiveX interfaces. A Web browser handles Internet protocols, supports the user interface, and provides a common look and feel for all applications. However, there are other environments, too, such as XML interfaces and wireless displays (cell phones and personal digital assistants), as well as older technologies such as 3270 screens (yes, they still exist!) and emerging technologies such as business portals and support for collaborative computing, heralded by Napster.
2. **Interface server.** The interface server delivers the content to a wide variety of clients using the

appropriate standards. Typically, Web servers use standards such as HTML that provide the functions necessary to put the Web pages up on the client's Web browser. Navigation between pages and initial data validation is typically performed on the Web server to reduce communications traffic and limit requests to the other servers. However, other protocols and standards must also be considered. XML has assumed a special significance, especially for specialized document passing tasks, and wireless computing standards are a rapidly growing area.

3. **Application server.** The application server provides a core set of technical services for scaling applications to the demands of large companies. This includes the ability to handle transactions, runtime persistence, security, and asynchronous communication across distributed systems. Business components are typically deployed within the application server. Indeed, one of the most interesting trends is the provision of business component frameworks "out of the box." The application server provides deployment capabilities and provides the necessary CEE. Different types of application server technology are now available from a variety of vendors. Microsoft's COM+/MTS, incorporated in its Windows NT and 2000 operating systems, is the most widely used. It is closely followed by BEA's WebLogic and IBM's WebSphere products in the EJB space. Other EJB-oriented products include the Oracle Application Server, Sun's iPlanet, and Bluestone's Sapphire/Web. In the CORBA space, Iona, with iPortal, seems to be the largest and best-known vendor (see the September 2000 *CDS* for further details).
4. **EAI server.** The EAI server manages consistency between existing systems, databases, and packages. It provides a focal point for all integration code, reducing the complexity of complex system integration scenarios, ideally using a component model such as CORBA to expose standard interfaces. Initial EAI products were based on proprietary standards, but lately many have converted to use XML as a standard interchange mechanism (read on for a discussion of XML). Different approaches exist. For example, connector technologies take the simplest approach to integration by providing mapping and translation mechanisms to allow different

systems and packages to communicate and share data; CrossWorlds (www.crossworlds.com) is one example. A message-oriented approach provides asynchronous communication among disparate systems based on management of queues; for example, IBM's MQSeries and the Microsoft Message Queue Server have been successfully used for building distributed applications requiring coordination of services from multiple sources. Other products, such as Vitria (www.vitria.com), integrate at the business process or workflow level, providing additional capabilities to messaging services for message brokering, intelligent message routing, and message translation.

The layers described above are very much abstractions. Although some products do fall neatly into a particular layer, products increasingly straddle different layers and offer differing capabilities. For example, Art Technology Group's ATG Dynamo product suite (www.atg.com) embraces a range of servers that include a Java 2 Enterprise Edition (J2EE) application server and several types of interface servers for personalization, storefront development, and content management. Many of the major industry players use the label "platform" to define the technical architecture offered by their products. For example, Computer Associates' Jasmine *ii* (www.ca.com) is an integrated e-business platform that includes application server features such as support for all the component standards as well as EAI connections, an OO database, and neural net technology known as Neugents.¹

The interesting thing to note here is the varying level of support for business components in terms of frameworks, tools, and layered services. For example, application server vendors are increasingly offering component frameworks, at the business level as well as at the technology level, that come shrink-wrapped with the products: for example, BEA's WebLogic includes component frameworks originally acquired from The Theory Center, and IBM's WebSphere includes an evolution of the earlier SanFrancisco component framework now known as Business Components.

¹In the interest of full disclosure, I am employed by Computer Associates as a principal component strategist.

Others, such as Xpedior (www.xpedior.com), offer suites of business component frameworks along with component management and execution services.

It's also important to consider interoperability capabilities between the different component models. As I'll discuss a little later, Web services are set to steadily cut through this problem. However, there are a number of already established routes, including bridging and broker environment bindings. The Sun and OMG server-oriented component models share much common ground, as evidenced by the CORBA 3.0 specification. Technically, the two are integrated at the protocol level using Java Remote Method Invocation over Internet InterORB Protocol. A degree of client-oriented Java/COM interoperability can be achieved using Microsoft's Virtual Machine. Options also exist at the server level; for example, BEA's WebLogic application server has support for static COM/Java bridging through its WebLogic COM component. COM and CORBA is perhaps the most challenging combination. OMG has created a standard for COM/CORBA interoperability, the COM/CORBA Interworking Specification, that defines mappings between the CORBA Interface Definition Language and ActiveX. There are several bridging solutions available based on this specification, including Critical Path's (which recently acquired PeerLogic) InJoin Broker COM2CORBA and Iona's Orbix COMet.

There are two broad problems that organizations face in the realm of integration technology. First, choice of products at the application server level is especially critical, and it is worth spending significant time and money on this decision. Although the theory says that a component developed to comply to one component standard is usable in any CEE supporting that standard, at present, vendors tend to implement CEE services their own way for their own products. Thus, it's currently tricky to port an EJB component from one EJB server to another.

Second, although things always move fast in IT, this is even more true with client environments and the supporting interface services. The emergence of different user channels, from cell phone to digitized TV, is likely to cause vendors to switch to support multiple interface

formats. Around 18 months ago, we were told that cell phones were about to revolutionize our buying habits. A couple of months ago, I ran an on-the-spot poll to see how many out of a technically sophisticated audience of about 250 attendees at *Component Computing 2000*, held in Finland in October 2000, had bought into this vision. One hand was raised — and this in the land of the mobile phone!

The Software Component Market

The vision of a software component marketplace is not new. The oft-mentioned software crisis was stated explicitly as early as 1968 at a NATO-sponsored conference, where it was recognized that producing software is so difficult it should be treated as an engineering discipline in its own right. It was expected that this would give rise to software assembly using software components. Prefabricated and tested building blocks such as integrated circuits, car parts, and switches were commonly used in fields such as mechanical and electrical engineering. Likewise, software components could be used as building blocks when developing new software, thereby reducing the expense and risk of developing everything from scratch. Just as component markets had been created in other industries, the same would happen with software.

Although much progress has been made since 1968, a genuine large-scale software component market did not start to form until recently. Estimates of market size vary depending on your definition of the market. If you're talking shrink-wrapped and off-the-shelf, the figure is US \$1 billion in 2002. If you include the full range, the figure is around \$33 billion in 2003. In other words, there are several aspects to the component market, and these aspects change. They change not only with respect to technology, but also with vendors' attempts to gain competitive advantage through branding and marketing-speak. I'll be reviewing this rapidly evolving market in a future issue. For now, I just want to make one or two observations.

Outsourcing of component design has been available for a while now through an online component marketplace. For example, Flashline.com provides a "Beans by Design" service that allows companies to post requests for specified JavaBeans that are then bid on by

developers (www.flashline.com/entrypage.jsp). More recently, CodeMarket (www.codemarket.net) was launched in a similar vein.

As far as buying and selling components goes, until recently, most commercial activity was centered on the trading of low-level small-granularity technical components. There are some companies that have been active in this space for a while; for example, ComponentSource has been in business since 1995 (www.componentsource.com). More interesting, the market in higher-grained business components is gaining momentum. Financial components developed by EDS and resold by ComponentSource have been available for a couple of years now. Other broker organizations are offering business component frameworks in vertical markets with value-added consultancy to help ensure the correct fit with organizational needs. For example, Objectools, based in Toronto, Canada, is offering various vertical domain components with consultancy (www.objectools.com).

On the direct-selling vendor side, there are similar moves. For example WebX, a London-based company, offers insurance frameworks (www.webx.co.uk). The emergence of early business components has been paralleled by the move, discussed previously, by application server vendors to offer component frameworks as part of their products.

Again, the industry has been talking for some time about a subscription-based approach to purchasing software. Thus far, this has taken the form of generic service provision at the level of ASPs hosting significant chunks of an organization's computing capability, particularly large software packages such as ERP solutions. Subscription-based Web services may be considered another aspect of this market that progresses toward a finer-grained, more flexible approach. I'll consider Web services separately in the next section; I predict they will be particularly significant.

Other organizations are seeking to exploit the middle ground of the subscription-based approach as component service providers. For example, Verge Technologies Group provides [ejip.net](http://www.ejip.net) (www.ejip.net/main.jsp), a hosting service for EJBs that lets users deploy their components and then make their services available over

the Internet. Revenue is generated from both hosting and usage fees.

Indications are that there are around 4,000 components in the market and that the number is doubling each year. However, when I talk to end-user organizations in my work as a consultant, I find that many inhibitors to this market still exist. Apart from the relatively low availability of business components, there are two factors at work here.

First, there is the basic question of "How do I know this is the right component?" It is vital to be clear on the definition of software assets and to achieve consensus on specification if the software marketplace is to really take off. The industry has a long way to go, although standards initiatives are under way. Another point is that nonfunctional requirements, always a problem in software specification, are even more of a problem with components because of the number of environmental contexts, each with potentially different nonfunctional requirements in which components may be deployed.

The second problem is that early component-management tools found it impossible to scale up. They suffered problems with consistency of the component definition that I've already mentioned, outside the province of well-scoped parts of single enterprises. Added to that, there are other challenges in enabling organizations to make their components available to a wide audience via the Internet, particularly with effective cataloging and browsing facilities. More fundamentally, managing software assets has always been a challenge. This challenge increases by an order of magnitude with components kept consistent in a repository. As the number of components, interfaces, and interdependencies increases, so does the need for good configuration and version management facilities.

Web Services

In June 1999, Microsoft emerged from its legal battles and apparent slumber to promote .NET, a major Web services strategy described as no less significant than the move from DOS to Windows. It is no surprise that since Microsoft's announcement there has been a deluge of proposals for Web services standards, with IBM prominently involved. What is perhaps more

surprising is IBM and Microsoft working together on these standards.

A Web service is a collection of functions that can be published, located, and invoked across the Web. These functions can be anything from simple requests to complicated business processes. Once a Web service is deployed, other software (including other Web services) can discover and invoke the deployed service dynamically at runtime. The mechanism used to do this is typically the Simple Object Access Protocol (SOAP), which uses XML to define the request's input and output parameters. Because SOAP is supported by all of the major vendors (and most of the minor ones, too), the pain of translation between CORBA, DCOM, EJB, and other protocols should be over. And because Web services can be written in any language, developers do not need to change their development environments to produce or consume Web services. In a nutshell, Web services are components — components for the truly distributed era. And like a component, a Web service can aggregate other Web services to provide a higher-grained functionality. For example, a Web service could provide a set of high-level vehicle purchase features by orchestrating lower-level Web services for vehicle selection, insurance, and warranty.

An early example of a Web service is Microsoft's Passport service, which provides customer identification and authentication (interestingly, this was originally labeled a "megaservice" by Microsoft in the pre-Web services days). Customers register their details once with www.passport.com and receive a user ID. The user ID, when used with Passport-compatible e-commerce sites, enables the vendor to retrieve all the shipment, billing, and other details from Passport without further customer interaction or the need to consult internal databases.

Toolkits from vendors like IBM and Microsoft are aimed at allowing developers to quickly create and deploy Web services. In addition, some of these toolkits allow preexisting COM components and JavaBeans to be easily exposed as Web services. In November 2000, Microsoft announced public availability of the beta version of Visual Studio.NET and the .NET Framework (<http://msdn.microsoft.com/net/scl.zip>), and IBM announced an early version

of its Web Services Development Environment (www.ibm.com/developer/webservices).

Meanwhile on 6 February 2001, Sun announced Open Net Environment (Sun ONE): an architecture and road map based on open standards, such as Java and XML, and comprising the Java 2 Enterprise Platform, Sun's Forte development tools, and iPlanet software. Though Sun places an emphasis on "smart" or intelligent Web services, it is very difficult not to see this initiative as a direct competitor to .NET; see www.sun.com/software/sunone/ for details of Sun's approach.

The vision of Web services is one of organizations building collaborative processes that flow seamlessly across the boundaries of business partners, providing visibility and integrity across the supply chain. Web services enable a "pick and mix" approach to the selection of value-added and commodity services. This service-based approach is very much the vision of component-based development that people like myself have been espousing for several years now: plug-and-play components that transcend traditional organizational boundaries and technical environments. My hope is that competing Web services initiatives do not work against this vision.

For Web services to succeed, there are many technical challenges that must be met. These are not new, and they apply to all types of components. For example, how do I know this is the right Web service to meet my particular need? Web Services Definition Language (WSDL) and Universal Description, Discovery and Integration (UDDI) are two emerging standards that are set to address this issue. Other questions center on non-functional requirements: How do you measure the reliability of a Web service provider? How do you know which other vendors to trust? How secure is the Web service? How are transactions supported? Does the Web service scale up? These are issues that we can be sure are being worked on right now as the major vendors vie for supremacy. For more details, see The Mind Electric's Founder, CEO, and Chief Architect Graham Glass' excellent set of tutorials at www-106.ibm.com/developerworks/webservices/.

Although the technical challenges are significant, perhaps the most profound challenge implied by Web

services is the paradigm shift away from layered architectures rooted in the days of client-server systems and toward truly peer-to-peer architectures. To date, the degree of interoperability achievable with component models such as COM+ and EJB has been compromised by proprietary runtime environments. Web services promise to break this mold by providing global interoperability. This is an evolving theme that is set to have a big impact on business-IT alignment. It reflects the move away from competing organizations that have fixed boundaries toward competing value chains of organizations with fluid boundaries. I'll be keeping an eye on the architectural impact of Web services on business during the months ahead.

Open Source

The open source movement has emerged in parallel with the component marketplace, again enabled by the Internet. In one sense, open source appears in direct conflict with the principles of components: code is no longer encapsulated, it's open for development! In reality, this is a gross oversimplification. For now, bear in mind that most open source efforts center on infrastructure software where the user is the developer. Examples include Sendmail, an e-mail routing program (www.sendmail.org); Apache, a Web server (www.apache.org); and Mozilla, the open source core of Netscape's browser software (www.mozilla.org). In this sense, open source represents another mechanism for developing software to the maturity of component status required by infrastructure software. Good techniques are therefore essential. For example, OpenAvenue is a company that is extending the open source model to collaborative development (www.openavenue.com). OpenAvenue has struck a major deal with Sprint PCS to supply its Oasis (OpenAvenue Source Infrastructure System) to its development groups. Oasis focuses on collaborative code development — for example, managing submissions and offering specialized code search capabilities. It will be interesting to see how the movement unfolds, especially with regard to the vertical application space.

Standards

Much of the work with standards is in a variety of areas working toward greater interoperability and interchange of information among enterprise solutions. I'll

be watching this closely as it unfolds. Right now, I want to draw your attention to some of the particularly salient ones.

OMG has moved well beyond its earlier focus on CORBA toward the Unified Modeling Language, the XML Metadata Interchange (XMI — OMG's XML system for integrating various models), the Common Warehouse Metamodel, a variety of industry-specific component/XML frameworks, and the Meta Object Facility, which, when combined with XMI, provides a truly abstract way of integrating a wide variety of specific models, including component models, visual modeling tool models, and database and repository models.

Rational Software's Reusable Asset Specification (RAS) centers on the classification, content, and usage of software assets (www.rational.com/eda/ras/preview/index.htm). RAS first came to my attention in the latter half of 2000 and is clearly a major initiative that is still unfolding. It is targeted at software in general, not at components specifically. In fact, it makes very little reference to components. However, it will be interesting to see how these standards can be applied to the fundamental question of "How do I know this is the right component?"

Distributed applications require interchange of data to provide integration and coordination between the various pieces. XML, a standard of the World Wide Web Consortium (W3C), is a tag-based language that can be used to describe information exchanged between senders and receivers and offers a standardized approach for defining these integration schemes. Its simplicity and the ease with which it can be extended allow it to be used in a wide range of integration scenarios in many domains. The XML document format embeds the information within tags that express the information's structure. XML also provides the ability to express rules governing the structure of the information. These two features allow automatic separation of data and meta data, allowing generic tools to validate an XML document against its grammar.

The big issue occurs in agreeing on the definition of the tags. For example, if multiple partners in an e-business are to take advantage of XML, they must agree on the semantics of business terms such as

“vehicle” and “reservation date.” Otherwise, the syntactical advantages of XML are lost. Not only do organizations need to standardize internally on business terms, they also need cross-organizational agreement. Many standardization groups are therefore working in different business domains attempting to create standard document type definitions to allow greater sharing among solutions within that domain. For more information, take a look at www.xml.org, hosted by the Organization for the Advancement of Structured Information Standards (OASIS), a nonprofit international consortium dedicated to accelerating the adoption of product-independent formats based on public standards hosts.

Microsoft is working to facilitate rapid adoption of XML through its BizTalk initiative, collaborating with several industry groups to create a set of standard XML tags. Its BizTalk server is an XML server aimed at encouraging rapid use of XML. RosettaNet is an XML-based initiative formed by a group of IT industry players to establish an e-business framework for IT supply chain integration. And OASIS has formed a joint task force with the United Nations CEFAC organization to create ebXML (e-business XML) as a basis on which the global implementation of XML can be standardized (www.oasis-open.org).

The limitation of XML is that it is data-centric. However, the strong industry initiatives that center on XML reveal a deeper significance: a rapid dawning that the interfaces of business systems are important. SOAP addresses this issue by providing a remote procedure call mechanism for XML that is independent of operating system or programming language (see www.w3.org/submission/2000/05/). SOAP doesn't resolve the conflicting claims of the CEE vendors in terms of whether to use CORBA, EJB/J2EE, or COM+ for the basic reason that a CEE is still required to implement components. What SOAP will increasingly do is provide interoperability between applications comprising different components using different CEEs.

SOAP looks well placed to provide connectivity between collaborating organizations (with differing technologies) along the value chain in terms of the Web services discussed earlier. The XML Protocol Working

Group (www.w3.org/2000/09/XML-Protocol-Charter) is working toward submission of the standard to W3C.

Three other key emerging standards involved in this field are worth noting:

1. WSDL is an XML-based specification schema for describing contracts between a set of services that are to exchange messages. It represents a merger of work between IBM and Microsoft proposed to W3C as part of a possible open Web services standard.
2. Transaction Authority Markup Language (XAML) is a language for managing distributed business transactions. A group including IBM, Sun, Oracle, Hewlett-Packard, and Bowstreet has proposed it to W3C and others (www.xaml.org). XAML complements WSDL by providing transaction management of Web services.
3. UDDI is an online means of consistent publishing and consuming of Web services that has been proposed by IBM, Microsoft, Ariba, and others (www.uddi.org).

Finally, and very briefly, the wireless aspect of the Internet revolution has just barely begun, both in terms of technology and the marketplace penetration of the existing technology. The Wireless Application Protocol is a fast-emerging family of protocols allowing mobile devices to access wireless services (see www.wapforum.org for more information). Just as XML semantic standards are emerging for different vertical sectors, I expect to see this trend for wireless communications. I'll return to this topic once things have evolved a bit more.

The great difficulty for user organizations is that standards change, merge, and fall into disuse. As someone once said, the beauty of standards is that there are so many of them. An example of this is the number of XML initiatives, even within common industry sectors. There is a constant tension between the efforts of standards consortia to unify common concepts and languages and the efforts of vendors to fine-tune and introduce proprietary features to gain competitive advantage. Witness the number of application servers supporting EJB but offering a range of different proprietary features to tune the environment. This situation is compounded by individual consortium members from

the vendor community wrestling to gain control of the standard, which is seen as the way to control the market.

Component-Based Strategies

Returning to the historical picture I painted at the beginning of this issue, practical strategies are needed to align use of the technologies with business needs, as shown in Figure 4. In a world of external partners and shared resources, enabling technologies that embrace standards are essential. They have the potential to accelerate time to market, integrate disparate applications, and ensure consistency and connectivity across the supply chain. The challenge is to embrace the enabling technologies and products (from a moving playing field) that have flexible architectures that align with business needs. At the same time, organizations need to adopt the technologies rapidly, without major infrastructural change.

Although the enabling technologies are still maturing and there are complex market forces at work, the state of the practice in many organizations falls well short of realizing the potential of these technologies. Sometimes the required strategies are completely absent. Instead, the organization relies on the super

programmer to dig it out of one crisis after another. In other organizations, top-heavy methodologies are too sluggish and fail to exploit the developments in technologies.

Despite what we read about e-business in the hip magazines, most large organizations are spending significant time and resources on their legacy systems. The money is not just spent on maintenance, either — the dramatic effect of Internet failures has taught us that it is vital to get the fundamentals right before you expose legacy functionality over the Web. Effective transition strategies must recognize this fact.

I'm not pretending this is easy; in fact, it's very tough — one client used the analogy of reengineering an aircraft while in flight, which I thought was on the mark. A delicate balance must be struck between the drivers of business and technology. I'll spend much of my time in future issues looking at the required changes in planning, architecture, process, design techniques, and organization guidelines. Equally, I'll be keeping an eye on developments in e-business and enabling technologies. Such changes require realistic plans to transition the company to an e-business as quickly as

- Market entry cost
- Product virtualization
- Diversity of channels
- Consumer power

- Architectures
- Processes
- Techniques
- Organizational guidelines

- Enterprise application integration software
- Application servers
- XML
- Web services

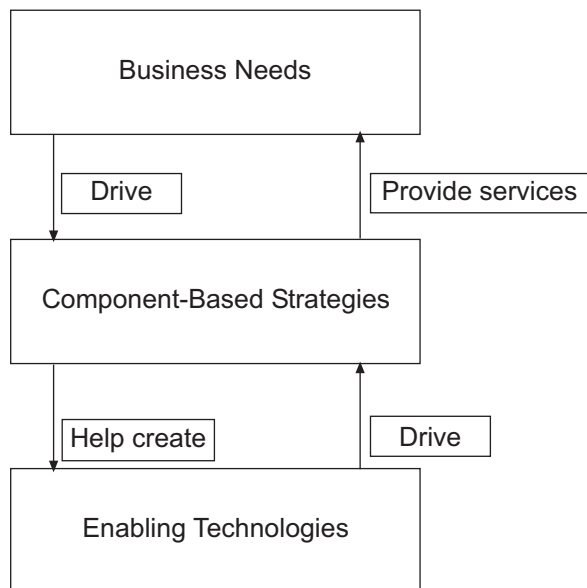


Figure 4 — Component strategies as a balancing act.

possible, while continuing to support today's immediate needs. Here's a quick look at some of the issues and challenges associated with this shift.

Organizational Structure

Although organizational structure is a subject in its own right, it's worth mentioning one particularly salient issue here: the emergence of a separate e-business department. Such departments have often sprung up as offshoots of marketing departments that have grown disillusioned and frustrated with the perceived failure of the IT department to provide fast Internet solutions. Staffed with a mixture of Web programmers and self-taught marketing folk, the e-business department rushes to implement Web sites that may look cool but that often fail to integrate with back-end systems. Business executives at a well-known bookstore that had set up a special e-business department to blaze a trail on the Web looked surprised when they heard that customers were unable to return books bought at the company's Web site. However, this should have come as no surprise: poorly integrated Web sites usually reflect poorly integrated organizational structures!

Business Process Improvement

I'm sure seasoned readers will be thinking at this point, "I've heard the argument for process improvement, better planning, and flexible architectures many times before. What makes it different this time around?" The reason is partly push and partly pull. On the push side, e-business is an increasing fact of life even for traditional brick-and-mortar enterprises. For example, there is a large hospital in London whose efficiency was severely compromised by the fact that on an average day it received 26 different truckloads of medical equipment for the hospital's 51 operating units. By working hard to understand and improve its purchasing processes in tandem with e-procurement software, the hospital reduced its deliveries to just three a day, with a corresponding increase in efficiency. The point to realize is that the process improvements that we should have done 10 years ago now simply *have* to be done.

Component-Based Architectures

On the pull side, component standards, integration technologies, and (especially) the advent of Web services are greatly easing previous constrictions of proprietary interfaces and platforms that failed to scale up. To

continue with a health analogy: the airways are starting to open up, and the asthma sufferer can suddenly breathe. But like an asthma sufferer, a sick organization needs the right medication and therapy (homeopathic or otherwise!). This means that architectural and infrastructure work that should have been done last time around now has to be done. Static enterprise models that ultimately end up sitting on shelves (or as Web pages) are clearly not appropriate. Component architectures must be made to work as change management frameworks that recognize there are fundamental and graduated changes that organizations need to make to progress. Increasingly, an organization can buy, out-source, or subscribe to the more generic features of a desired solution. It can then focus its resources on the more specific features that make the solution truly valuable to the customer. Component architectures need to help with this kind of approach by exposing component interfaces and their dependencies.

Component-Based Processes

In the old days, we were faced with a simple choice: build or buy. The options were pretty clear. Although the decisions were not always easy, we could at least make some well-reasoned tradeoffs. The rapidly evolving integration technologies and expanding component marketplace have changed this picture dramatically. There are now an increasing diversity of different options for components and services that can be mixed and matched in various combinations from simple component purchase to Web services subscription and component framework extension.

At one extreme, it's not too difficult to imagine a scenario in which developers react to the increasing pressure of deadlines by delivering thousands of Web services just because they can, without heed to business needs. The call for clear architecture and process guidelines magnifies with the opening up of the software playing field. At the other extreme, today's mainstream software processes — iterative, incremental, or otherwise — reflect a bygone age in which systems were developed from scratch. This is ironic: e-business systems aren't really "systems" at all; they're sets of interacting components from mixes of sources, as our discussion of enabling technologies has shown. The "system," if there is one, is never really complete. The

imperative is to move toward lean adaptive processes that reflect this everyday reality — your survival may well depend on it!

Techniques

Changes in technique are also required. I'll briefly consider one example. Gap analysis involves making tradeoffs between what the business would ideally like and what it's realistic to expect. For example, an analyst might reuse the interface that "comes closest" to the ideal. In fact, sometimes it may be that things are compromised so much in the interests of saving money and time, by reusing what's already in the bag, that the original business process is significantly changed. Okay, it's not perfect, but maybe it's cost justified and practical to settle for less-than-perfect business process and perhaps upgrade it later by rewiring in interfaces that better match the ideal.

Cultural Issues

The technical and methodological challenges are tough, and there are a plethora of associated cultural issues that are even tougher. Again, I'll just briefly look at one. Although software reuse has been a Holy Grail of software development for a long time, because it is such an attractive concept to hard-pushed IT managers, treating reuse as a goal in itself leads to a range of problems, particularly cultural barriers such as the not-invented-here syndrome. I'm reminded of one organization that saw fit to run a Reuser-of-the-Month competition. In contrast, successful organizations treat reuse as a means to an end, of better solutions that ultimately result in more efficient and competitive business. Again, this calls for clear process guidance.

Summary

Developments in e-business and advances in component and Internet technologies require corresponding shifts in development practice that have been slow to emerge. In particular, it is the very process of software, so deep-seated in the IT psyche, that needs to change. At the same time, components are not a panacea for all ills. In one sense, they raise an entirely new set of issues. For example, how does a consumer know that a component is the right one? Where are the techniques to enable suppliers in the component market to

collaborate? Does this particular technology lock me into the vendor, or is it really as open as claimed? We struggle to find the answers in textbooks and Web sites — traditional, fast-track, or otherwise. Yet these are the issues that I come across in many organizations today. Over the coming months, I'll be seeking answers to these kinds of questions, reporting on experiences in the field, and keeping an eye on related developments.

About the Author

Paul Allen is editor of *Component Development Strategies* and principal component strategist at Computer Associates. He is widely recognized as a thought leader in component-based development (CBD). He specializes in the areas of business-IT alignment, software process, component modeling and architecture, and e-business transition management. Mr. Allen's detailed knowledge combines with a uniquely practical understanding of the problems companies face as they begin to develop enterprise e-business systems. His pragmatism stems from 25 years of experience in the development of large-scale business systems. Among his many roles, Mr. Allen has worked as project manager on many large commercial systems, as senior methods advisor to a major telecommunications company, and as consultant manager with Yourdon, Inc. He was vice president of methods at Select Software Tools, where he was a key player in shaping and implementing the company's CBD vision in the shape of the Perspective method, before joining Sterling Software (now part of Computer Associates) in April 1999 to lead its CBD practice.

Mr. Allen writes frequently on CBD and e-business and is a popular speaker at industry conferences worldwide. His coauthored book *Component-Based Development for Enterprise Systems* was one of the first and most practical descriptions of what was involved in building component-based applications. His latest book, *Realizing e-Business with Components*, was published by Addison-Wesley in October 2000. Additionally, Mr. Allen works regularly with industry bodies to help review software practices, ensure standardization, and develop practical technology migration strategies.