
# Web Services

## Helping to Transform Government

**Uttam Narsu**
*Vice President, XML and Web Services*

**Presented at Federal CIO Council Meeting, October 11, 2002**


**Technology Advice.**
**Business Results.**

Giga Information Group®

# Tremendous Interest in Web Services

- Initially hyped, now anti-hype, yet:
  - Government interest remains strong
  - Businesses merely deferring expectations of its value
- Recent CIO surveys:
  - Application integration (including Web Services) #2 priority behind security
  - 34% already have Web Services strategy in place with 28% planning on implementation in next two quarters

# Key Concept: The Service

**Reply (to Request)**
- **Message**
- **Transaction**
- **Business Unit of Work**


**Request**
- **Message**

**Service Interface**

**Legacy Apps/ Components**

**New Services**

**New Components**

It is important to draw a distinction between a service and a component. A service may just be an additional interface to a single component, in which case the component and the service can be said to be semantically equivalent. But to think of services in this way would be to miss the point, because if all the low-level fine-grained components in the world simply get turned loose on trading partners through Web Services, the world will be a very complicated place, and needlessly so.

No, the true promise of services can be delivered only if service interfaces and the resulting interactions are specifically designed to provide an interface that is convenient to use from the point of view of the service user, that delivers its services in a way that is as closely aligned with the specific business request the user wants to make as possible. This is rather like a drive-through window of a restaurant: you just drive up, place your order, pay, and get your bag of food. The bag is like the reply to a service request, a specific message, transaction, or business unit of work. The food inside is like the underlying components or application interfaces that implement the service.

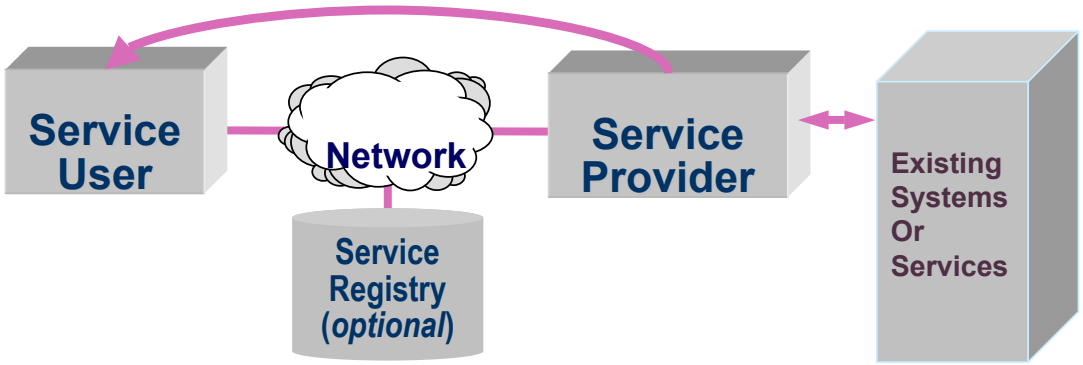
When you go to the restaurant, you don't want to have to instruct the fry cook to make the chips or French Fries, or the burger-flipper to make the burger. You want a convenient layer of service-interaction "glue" as provided by the person staffing the drive-through window, to handle those details for you. Well designed software services will offer this same kind of glue, convenient syntactic sugar to ease usage of the service, and also taking into account the need for packaging and aggregation of lower-level application calls, components, and interfaces. Additional design goals include optimizing for performance over slower network links with longer latency (not good for chatty component interface invocations), integration into an implicit workflow with known state, and enforcement of quality-of-service agreements that need to be monitored and controlled.

# What Is a *Service*?

**A *Service* (or *e-service*):**

- **Has a described interface (*may* be self-describing)**
- **Providing access to a software application service**
- **Usually with some larger-grained business value/function**
- **Provided via a defined network access mechanism**
- ***May* be dynamically discoverable**
- ***May* be hosted "elsewhere," with defined QoS, SLA, etc.**

**Alternative Service Identity Credentials Exchange**



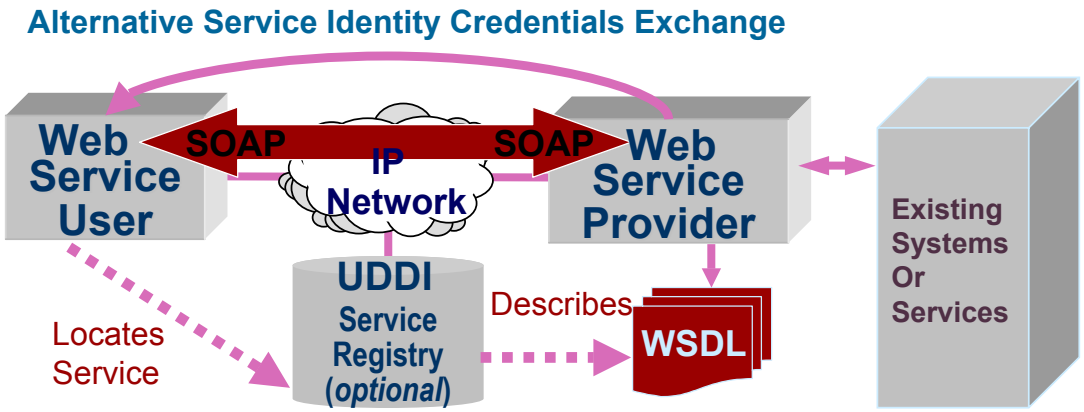
Viewed in a more general way (not specific to a particular technology implementation), a service in the application software domain, sometimes also called an *e-service*, is a unit of business value or function that is hosted by a service provider, delivered to a service user through a described interface over a defined network access mechanism agreed between the service provider and user. The interface may be self-describing, may be dynamically discoverable, and may be hosted by an environment distinct from that of the service user (but users and providers can even be in the same box if desired). And services are often just a front end to an existing application system or service implemented in a different technology.

Note that many current uses of the concept of services don't register the service for dynamic discovery, because the service is used internally or exchanged among a small group of trading partners that already know and trust one another, and they have agreed some other way to exchange the service identity credentials required to access the service. These alternatives may indeed be essential in the absence of standards to ensure the security of service access.

# What Is a Web *Service*?

**A *Web Service*:**

- **Has a Service interface implemented with SOAP/XMLP**
- **With SOAP/XMLP likely implemented over Internet protocols like HTTP and/or SMTP**
- **With the interface described using WSDL**
- **With the service registered/discoverable via UDDI, if at all**



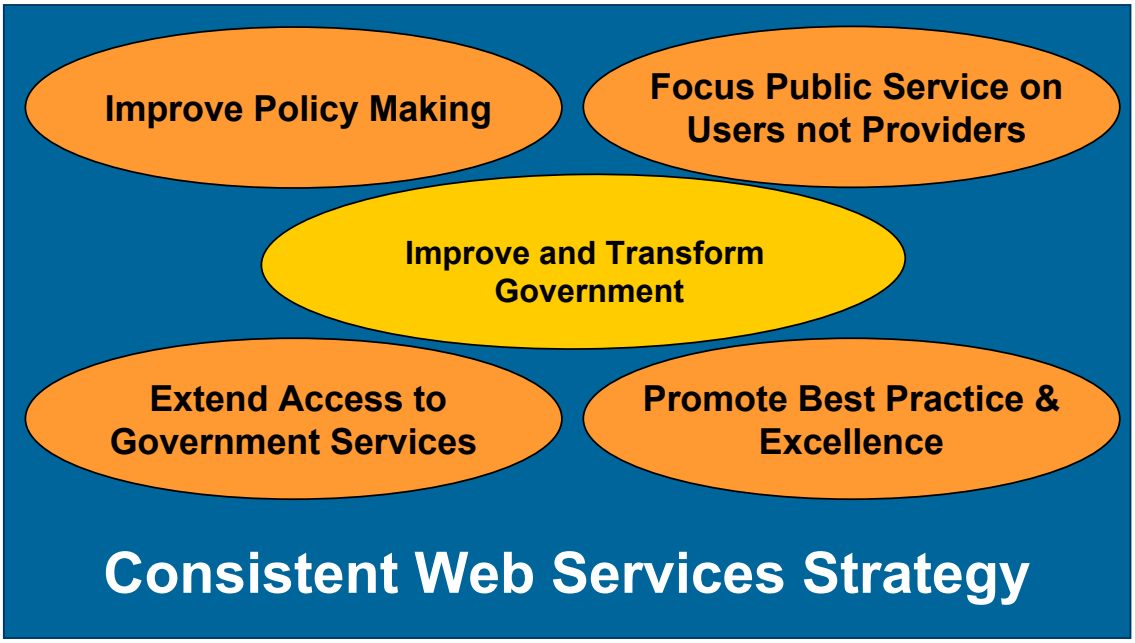
So, a Web Service is the specific implementation of this concept using Simple Object Access Protocol (SOAP), which may be called XML Protocol by the W3C, where SOAP is also likely to be implemented over Internet protocols like HTTP and/or SMTP. Yet this is not a requirement, SOAP can still be SOAP and use some other protocol underneath.

A Web Service interface is described using Web Services Description Language (WSDL), which is a form of XML document. And if the Web Service is to be registered and discoverable, this is done via the Universal Description, Discovery, and Integration (UDDI) interface. But even in the specific Web Services context, some Web Services are either registered only in private registries accessible only to one company or on an extranet to a specific group of trading partners, or not registered at all, with credential exchanged in some other way, even as simple as e-mailing a WSDL file and URL to call to a trading partner after they have signed up for the service.

A definition of a Web Service is:

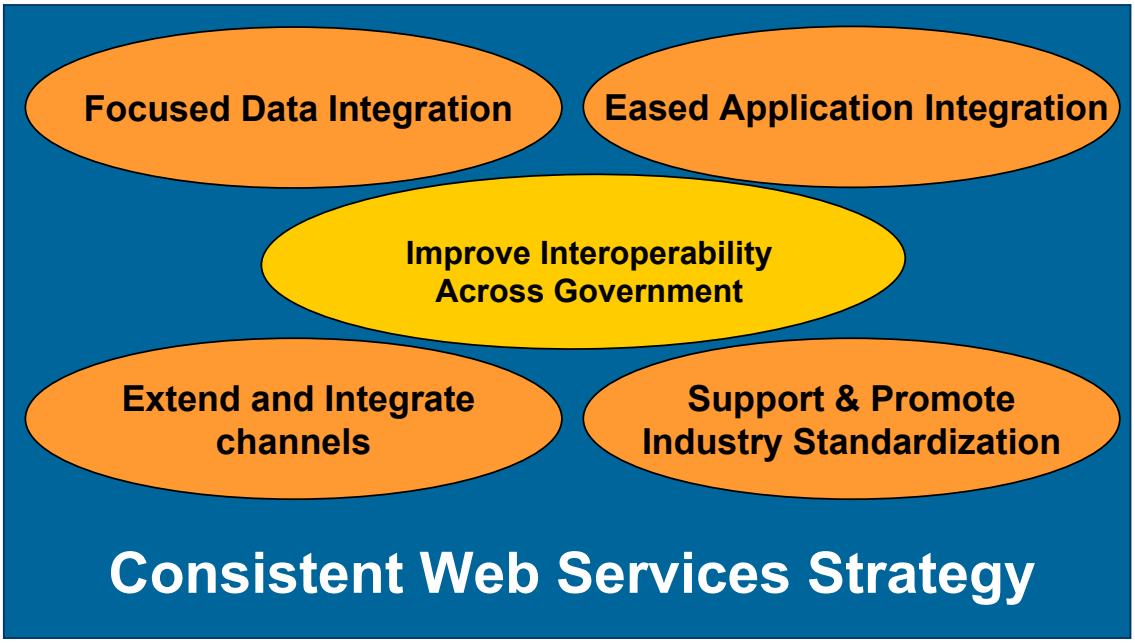
> A Web Service is an interface implemented by one or more applications or components that provides one or more business services to other applications or end-users via standardized Web protocols.

# Business Benefits of Web Services

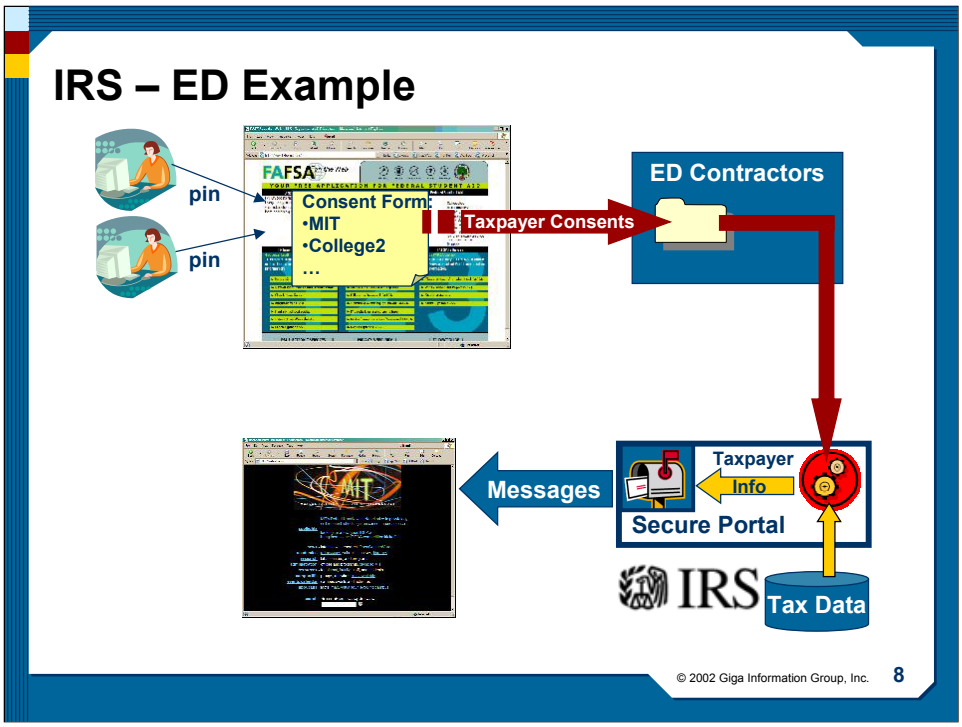



Source: Giga & GovTalk

# Technical Benefits of Web Services




Source: Giga & GovTalk



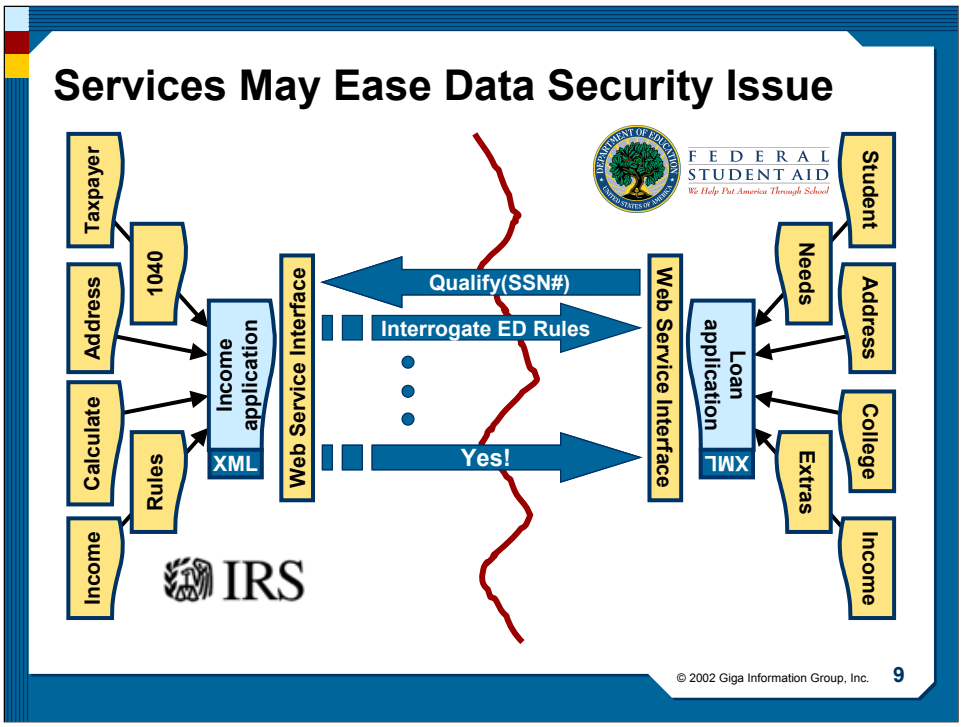

A student, spouse, parent/guardian (in the case of a dependent student) will complete a consent form on the Department of Education's Free Application for Federal Student Aid (FAFSA) website. The student authorizes the IRS to release the student's tax information to one or more colleges/universities listed on the consent form. They electronically sign the consent with a five-digit, self-selected PIN and authenticate themselves on the form by providing their date of birth and adjusted gross income for any one of the past three filing years. (A manual pilot with 8 colleges/universities is targeted to begin 10/7/02-ED's web site is ready. IRS employees will manually access and print the consent for servicing with the current product. Pilot will consist of 600 requests)

Mid 2003, ED and ED's contractors will be registered users of the IRS' e-services Transcript Delivery System (TDS) program. ED contractor will bundle together the student consents and electronically submit them to the IRS via the IRS' secure internet portal on a periodic basis (hourly or daily).

TDS is designed to provide a tailored transcript of specific information to will assist ED's colleges in determining eligibility for federal student aid. TDS will authenticate the student by corroborating their name, social security number, the date of birth and the selected adjusted gross income. Once corroborated, TDS will retrieve the income information for the student and place it in the secure mailbox of the designated college/university.

Representatives of the college/university will also be registered users and have access to a secure mailbox on the TDS system. When TDS places information in the secure mailbox, an e-mail is sent to the school advising them to check their mailbox for information. The college/university users retrieve the information and determine the student's eligibility for federal financial aid.

If there are any problems in retrieving information from TDS (e.g., return information cannot be found or the student's authentication information ), a letter will be mailed to the student advising that their automated request was unsuccessful.




The example that was posed requires some fairly complex information flows. Now if we multiply this scenario over several times with other agencies, and also factor in the increased concerns about privacy, and security after 9/11, then the requirements for legal intervention by Congress become burdensome.

To some extent, Web Services may be able to mitigate the data security/data integration requirements if we assume that some departments like ED can surface their unique business rules (i.e., who qualifies for aid) as Web Services.

Then ED can ask the IRS (on behalf of the colleges and itself), by sending a Web Services message: "Does (SSN#) Qualify for Aid?. The information flow would then be:

ED ➔ Does SSN#123 qualify for student aid? ➔ to IRS

IRS ➔ interrogate ED rules, wrapped as services ➔ to ED

IRS <evaluate / calculate>

IRS ➔ Yes / no ➔ to ED

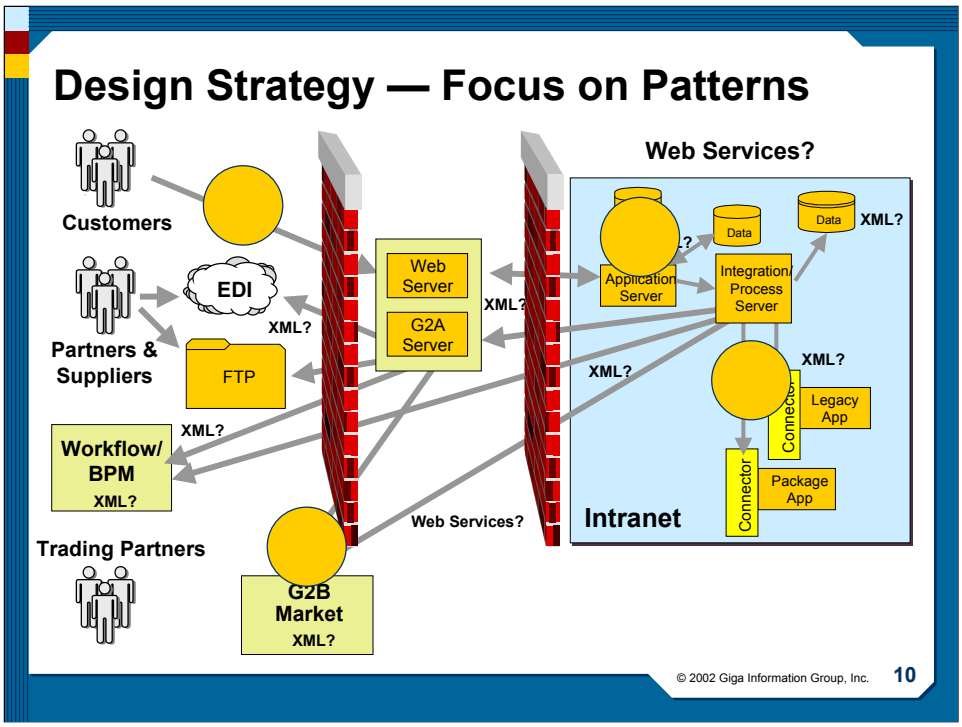
A response of Yes or No, mitigates the issue of ED needing to know income data (and the thorny issue of ED having to manage that info).

Business rules are being standardized in XML. Sending an XML representation of a business rule to be executed by the IRS in a DMZ, is an alternative (and more general approach to the above problem. Sending business rules as XML has the value of reducing the exchange/interrogation, is therefore more secure, and rules engines are becoming standardized. The new information flow with a rules approach would be:

ED ➔ Does SSN#123 qualify for student aid with **these** rules? ➔ to IRS

IRS <evaluate / calculate>

IRS ➔ Yes / no ➔ to ED




As one can see from the typical view of an n-tier application, the number of possible places where XML and Web Services can be constructively deployed is quite large.

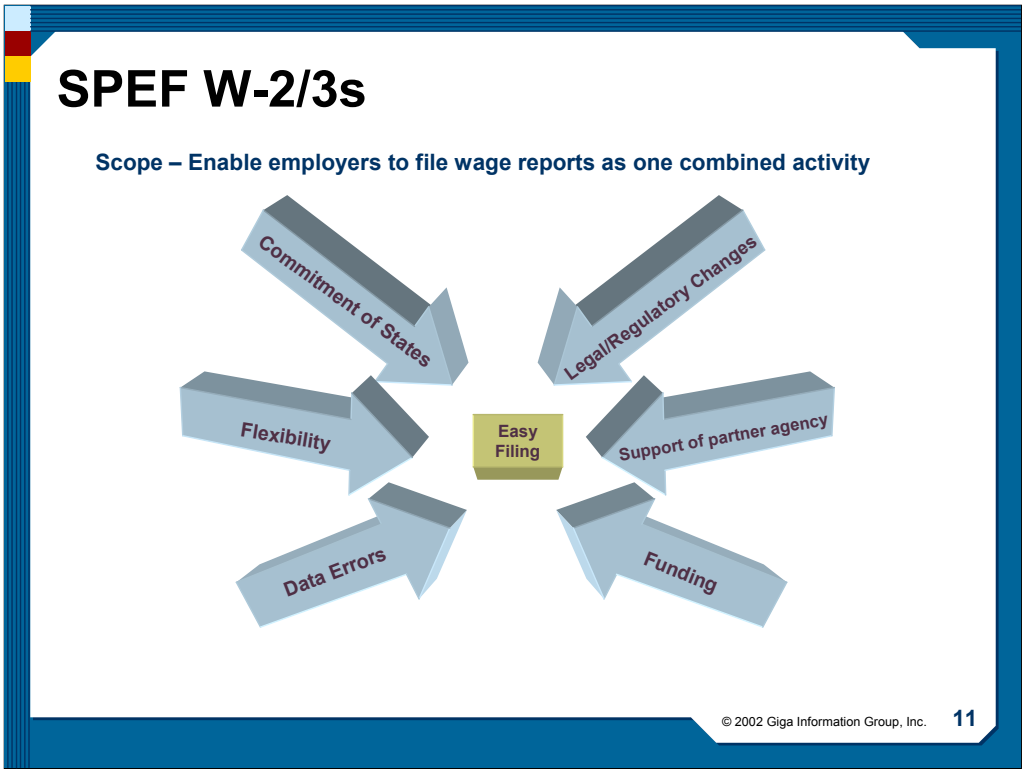
Never blindly add XML everywhere. Deciding to where to use XML can be a complex decision because one has to balance functional and performance needs against the inherent flexibility that XML adds. A functional need for evolving application configuration data, for example, doesn't require XML, however, using XML may allow one to evolve the configuration data format more gracefully.

XML's performance impact doesn't merely affect the network; larger messages mean more CPU time to parse and unparse messages, increased storage requirements (here the tag-to-text ratio is very important) and increased memory usage (especially if DOM-based parsing is used).

Similarly, a Web Services interface should emerge from business requirements, not from a decision to take an existing interface and "expose it" as a Web Service.

The key to accomplishing the above is through the use of patterns and guidelines, which will help manage the complexity and decrease the risk of adoption of XML and Web Services. Several industry patterns, such as Multichannel Interface, Spoke and Hub, Canonical Data, etc., are depicted above.

Focus on identifying patterns in the design and application strategy.




To reiterate:

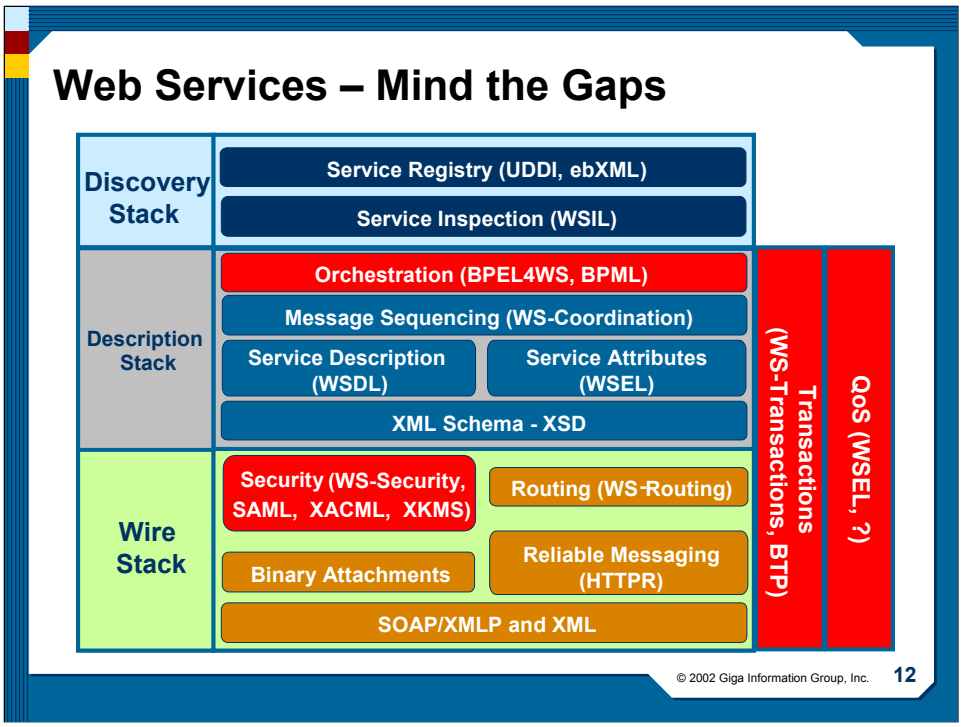
- Need all the states to be successful; otherwise won't reduce employer burden. Could be interpreted as infringing on states' sovereignty.
- States would have to designate the federal government as a single-filing point for W-2s; would require legislative or regulatory changes.
- Data problems significant barrier. Multifaceted (e.g., data transcription, data completeness, data errors in companion legacy data bases {SSNs}, multiplicity of data definitions between intergovernmental entities, improperly submitted data , data timeliness).
- Resolution of some errors would require regulatory or legislative action by both federal and state governments.

These are deep problems and it would be impolitic, and simple-minded to suggest that Web Services could, silver-bullet style, overcome these issues. Yet, Web Services may make some of the inherent problem difficulties more tractable.

The commitment of the states is in part due to deep integration concerns, and also fear that any imposed solution would result in a loss of control. At its heart, this is a variation of a classic integration problem, overlaid with concerns about local data control. I know of cases (Fidelity, DG Bank) where the corporate world has faced similar challenges and used Web Services with its promise of loose coupling, and XML's ability to tolerate multiple data schemas coupled with the transformational capabilities of XSLT promise to reduce the impact of this concern.

Similarly, Web Services layer of technological abstraction, which enables change in the implementations while agreeing on the interfaces could enable easier support of partner agencies, at potentially dramatically reduced cost.

Data problems in part could be surmounted by (a) not moving the data around(!) as much (b) through transformation and (c) through encapsulation of business rules



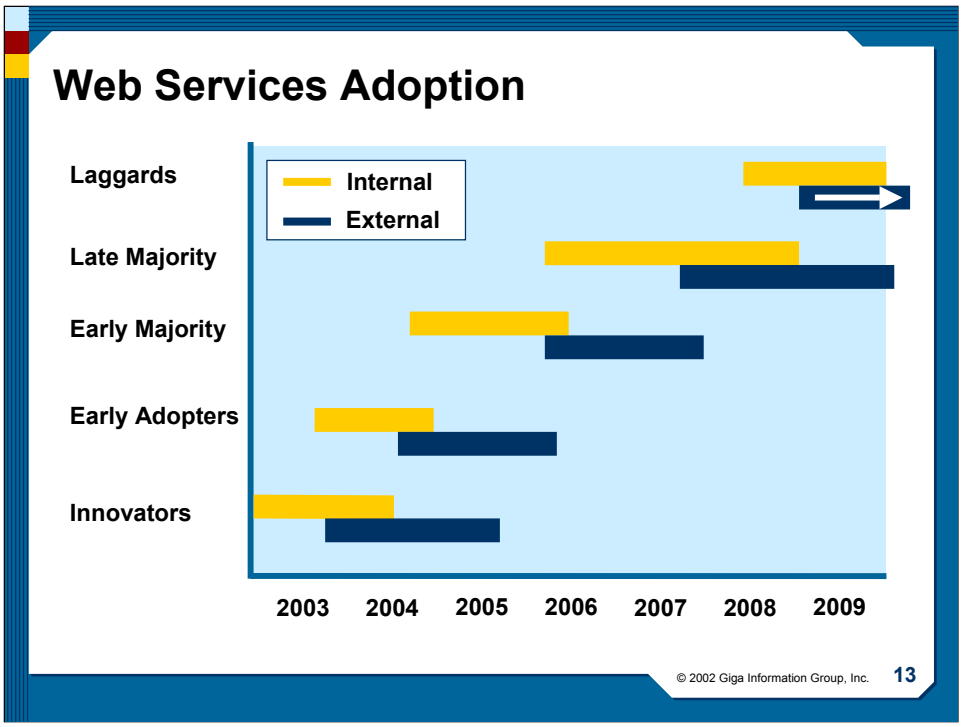

During the last six months, the concept of a logical Web Services Architecture (WSA) has taken root. Proposals from IBM, Microsoft and several other vendors are converging on a common set of standards-based services that such an architecture must provide.

(Note that the presentation layer is not visualized here to preserve some simplicity! Proposals for the presentation layer are undergoing development with much work from OASIS.)

General consensus highlights the need for between 15 to 20 Web Services standards. Today, there is only widespread agreement on three: SOAP (and the W3C standardization of it, tentatively called XMLP), UDDI (backed only by an industry consortium, not by the W3C), and WSDL (which was submitted to the W3C, but has seen little activity in the last year). As you can see by the slide, there are several vendor proposals and proposals that are associated with other standards bodies like OASIS and IETF.

While consensus around a Web Services Architecture will take years to emerge, we urge you to create one, independent of any application architecture you have. Creating your own Web Services Architecture will be the best approach to balancing the immediate need for a platform-neutral integration layer, with the long-term need for managing its change and growth in alignment to business strategy.

While many of the “boxes” can be filled in today, the ones in RED (WHITE in a black and white printout) represent gaps that are critical to the success of Web Services to address all of the use cases discussed on slide 14). We discuss them in passing here, but they are covered in more detail on www.gigaweb.com.



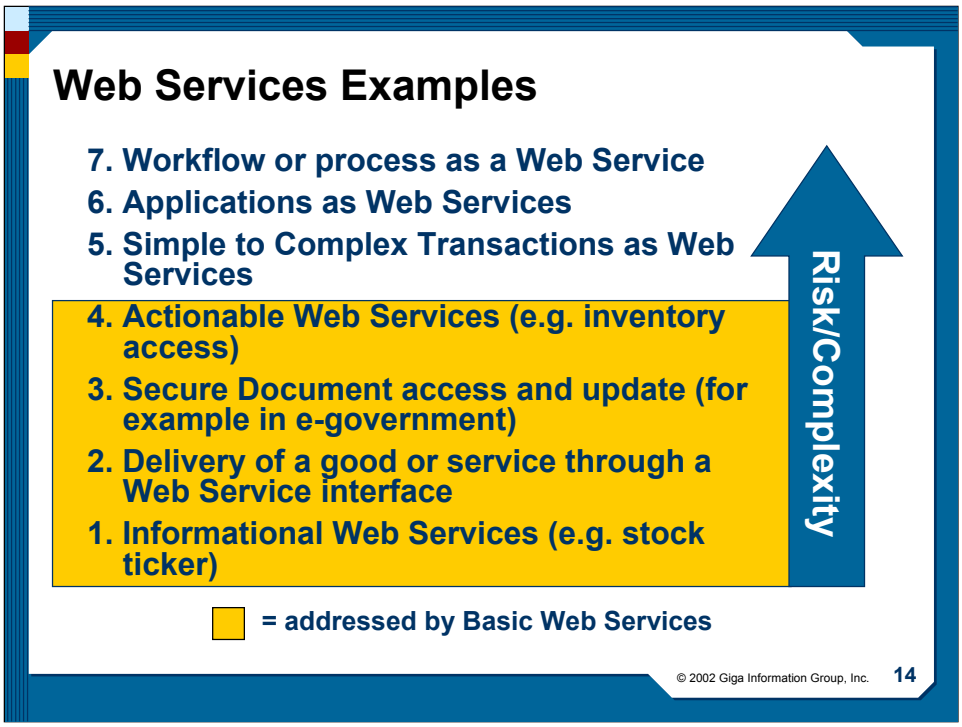

For some organizations, simple informational Web Services will be the first deployed, especially if the demands for security, and demands on the underlying infrastructure are light. For these companies, this will take the form of replacing a previous generation of Web-based technology (server-side scripting, CGI and the like). Others will have doubts that there is business value in replacing a technology (server-side scripting) that works.

For these organizations, creating access paths to stove-piped applications will be the compelling driver behind Web Services. Still others will see Web Services as a means to pursue an XML-EDI solution that will enable smaller suppliers to participate in a value chain. In other words, it is difficult to generalize about Web Services adoption patterns, since they will be largely driven by business need, vertical industry and the adoption profile of the organization toward technology.

We forecast that the most prevalent use of Web Services over the next 5 years will be as a standards-based approach to application integration — first internally, then with carefully chosen business partners. For the present, Web Services will not reduce the demand for integration servers as those servers clearly provide additional value in their support for mapping, routing, transformation, and publish/ subscribe, Web Services, however, will transform some aspects of the role these servers play.

Given these points, Giga predicts that the “Innovators” and “Early Adopters” of Web Services technology will develop and deploy Web Services for internal use in the 2002 to 2004 timeframe. In 2004, the “Early Majority” join the fray, and in 2006 the “Late Majority” followed by the “Laggards” several years hence. Because of the technology issues with security and messaging, external services follow internal service usage as depicted above.

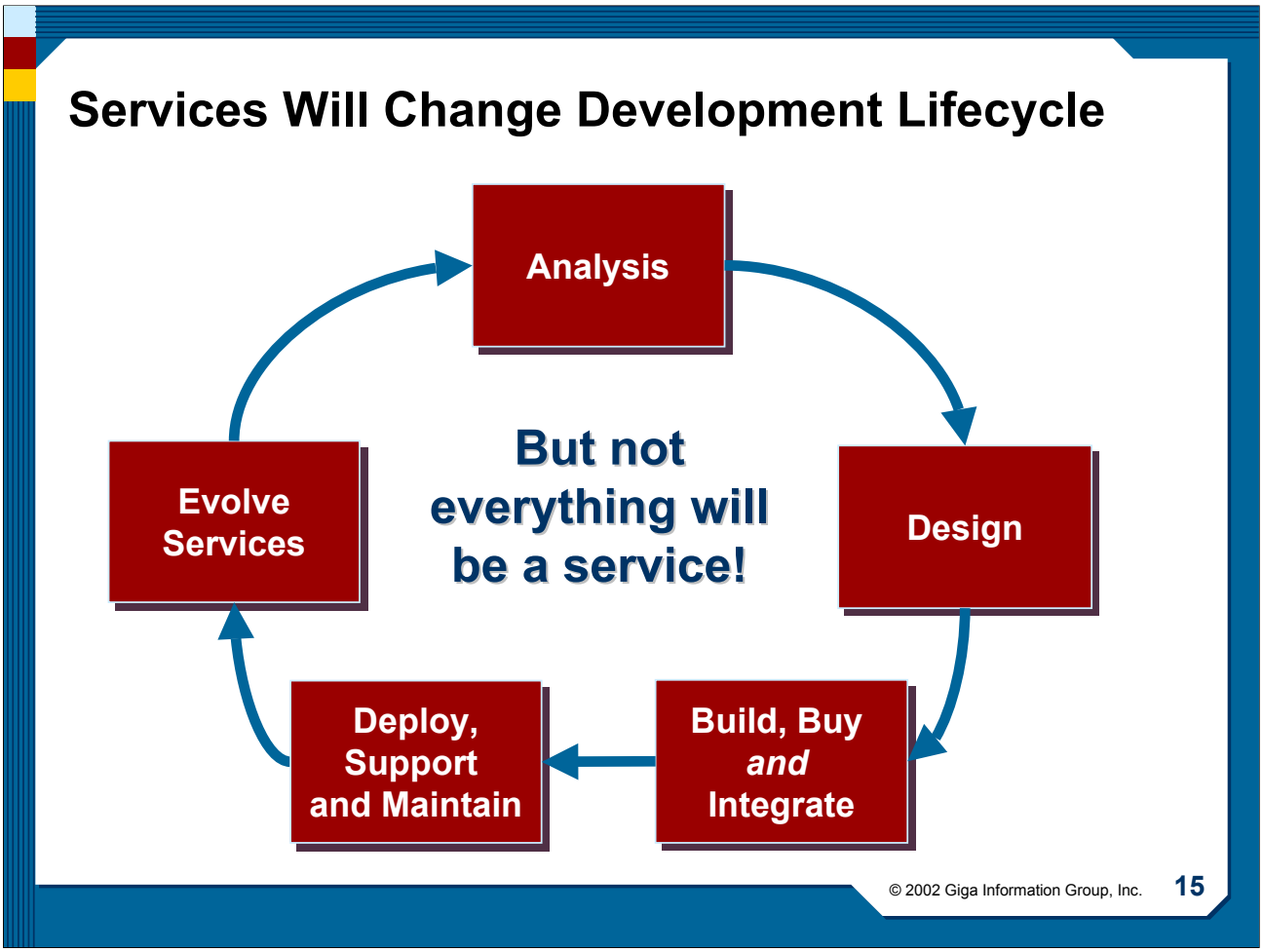
Know who you are (early adopter vs. mainstream, etc). Consult what others in your vertical are doing. When to start preparing depends on the state of your IT environment and your time frame for implementation as indicated above. Since business demand for functional interoperability should be the driver for Web Services projects, the most likely candidate applications are those that the business has been eyeing for some time with the hopes of modernizing and integrating with newer applications. Yet IT-driven demand for Web Services may also be valid to drive small proof-of-concept (POC) projects that flatten the learning curve for future applications development. In well-funded IT organizations, these POC applications can begin immediately, while other organizations will wait until the economic climate improves.




With the above limitations, it is no wonder that organizations are very cautious about where and when they first deploy Web Services. It seems reasonable to assume that adoption will follow the path taken by classic Web technology, along a continuum from low-value, non mission-critical, to high-value, mission-critical. The slide depicts examples of Web Services, from lower complexity and lower risk, to higher complexity and higher risk. It's worth noting that these examples can potentially be either internally or externally deployed Web Services; for the case where it's external, the need to provide security raises the risk and complexity level over the internal form.

Service-based design identifies and rationalizes candidate services which must be identified and rationalized with the business objectives. Not every function will be a service, nor should they be. In theory, a service boundary should align with commonly needed functions, represent a complete unit-of-work, and have a clear and communicable end result. In order to be good candidates for reuse, the functions should be in fairly constant demand. Examples of simple services may return a customer balance, credit-status or order-status — each are valuable in many applications and return a fairly discrete answer. Reporting functions can also be good early examples of services that provide business value, yet carry little risk.

With candidate services identified, examine the underlying technology of those applications — are they stable and mature, is it possible to wrap the underlying application functionality as components so that services can be layered cleanly on top? How do the standards tap the existing systems — do the applications have a viable API, can they be accessed via the presentation layer using Web-to-host tools, or is application mining a viable approach? How large a factor is transaction latency? A tactical adoption approach minimizes organizational and process change in favor of quickly adopting Web Services for high-value business needs. A strategic approach will focus more on the transformation of the organization and process, to accommodate a services-orientation — with each line of business analogous to a service bureau. These service bureaus identify their business "plug points" — where they naturally interact to exchange discrete information. The plug points and their relationships equate to service interfaces that will enable the business to reconfigure itself. Adopting a services-orientation requires you to identify the cohesive business interfaces that constitute a service as well as the loosely coupled relationships between the service units. Successfully adopting a strategic approach will require a strong architectural focus.



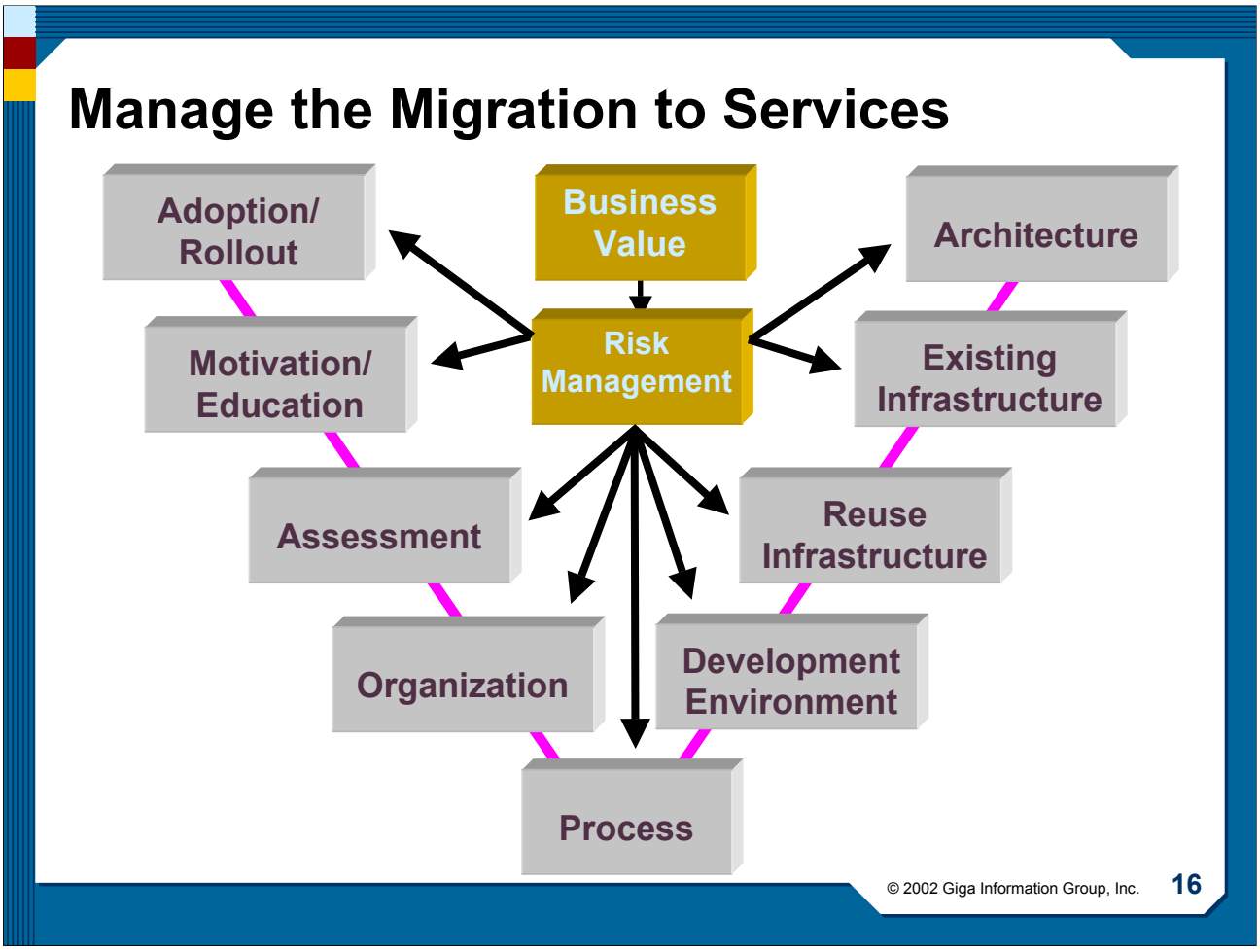

The key point is that not everything will be a service. So Services don't demolish everything that went before. They are yet another wave that organizations must surf to maintain agility. The picture above depicts the changes necessary to the classic application development life cycle to accommodate Services, including their integration and deployment. This life cycle must change in several ways:

- New deliverables and design concerns
- New stages and testing considerations
- New stakeholders/constituencies
- When and how to use, or not to use at all

The impact on development methodology is no less subtle. In the last five years, traditional systems analysis has withered under the onslaught of design-centric approaches that tried to bridge the gulf between analysis and design. Services and their complex interactions may make the case for a more analysis-centric approach to Use cases and scenarios.

Designing the interactions for performance and other operational considerations will likely require special knowledge about network and security issues, as well as a low-level understanding of how requests are packaged and what overhead is introduced by various packaging options, as well as the design trade-offs in their impact on development complexity.

It will be essential to ensure that the application architecture is aligned with the integration strategy, based on the need to implement a service-oriented architecture.




The **adoption** package organizes the high-level team entrusted with overseeing the adoption of service-based development (SBD) throughout an organization. The resulting plan and business case will establish a detailed organization and external consultant staffing plan, a **rollout** plan, and identify critical success factors. The results of this initial package will ensure that all stakeholders and participants understand what is required for success.

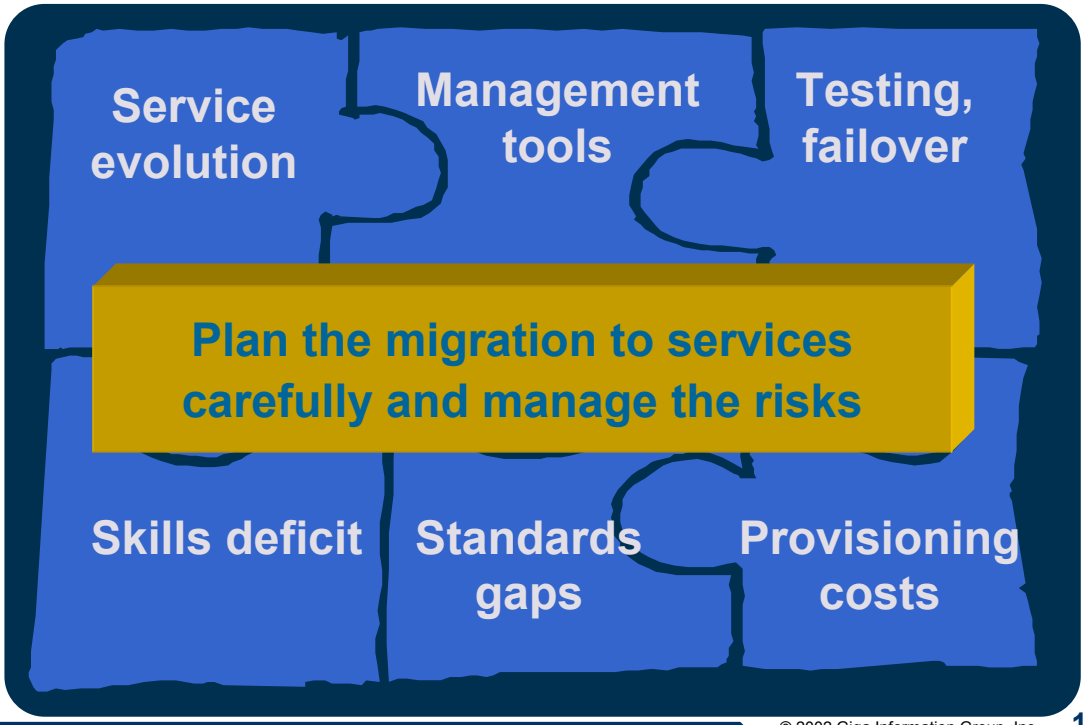
The **motivation/education** package provides motivation for and education about SBD. The **assessment** package's aims are to ascertain the "as-is" state of an organization's development organization: it's processes, roles, methodologies, organization, best practices and technology. Transitioning to a new style of development requires understanding the old ways of development and the unique organizational culture of an organization in order to understand how to formulate the adoption, education, architecture and methodology to be employed for SBD.

The **organization** package identifies any high-level or organizational changes that are necessary to enable the migration to SBD. The **architecture** package identifies the architecture standards to be used for SBD development. The **existing** infrastructure package identifies the existing standards in place that will be used for SBD development. The **reuse** infrastructure package identifies what technical and process infrastructure needs to be installed to foster reuse.

The **development environment** package specifies what programming languages, modeling tools, development tools and testing tools are best-suited for service-based development within an organization.

The **process** package specifies what development processes are best-suited for service-based development within an organization. SBD has at its core a different process than standard development because it must fundamentally allow buy vs. build decisions at any time after requirements definition. Also, SBD does not have a standard development process associated with it, and vendors are proposing different software platforms to help enterprises migrate to SBD. The optimal approach for an organization is to develop a customized SBD process for development because there will be less paralyzing change to the existing ways of development. To protect against a future standard process emerging, this custom SBD process must be built on standardized notations, must adapt standard processes where possible, and must document differences where they exist.

# Don’t Forget the Risks







There is little data available concerning the relative costs of evolving services in service-based development (SBD). Certainly, costs will be higher and there will be supplemental costs associated with: finding and qualifying a service, negotiating, managing and tracking service licenses and supporting dynamic evolution of a SBD system. XML standards are very immature for both Quality of Service (QoS) and service evolution.

Another risk is in the area of management and deployment tools for services, which are just emerging and are not very mature. Early leaders in this area are not the mainstream platform vendors, but are startups like Grand Central and Flamenco Networks.

Properly planning for and testing services will be challenging and may require every service provider to provide a separate test environment, including processes and policies for managing the environment, or for scheduling use of the test environment. For debugging support, the provider must at minimum have support staff available and may need to build debugging functions into the e-service.

Skills deficit is a standard risk, but no less real. One factor that acts to mitigate this risk is the incremental nature of service-based development. It really isn’t that far form traditional CORBA, DCOM, messaging and J2EE development. XML skills are increasingly available, but still at a premium compared to platform development skills such as J2EE.

The issue of standards gaps is a critical issue. Several recent Giga polls have highlighted this concern as the leading, or second leading cause for concern. This issue is detailed more thoroughly in the presentation “XML Standards and the Application Platform”

# Define a Web Services Strategy

- Wrong Approach? Like an ‘SQL Strategy’ instead of Database Strategy
- Yet, disruptive technologies require strategies
  - Focuses on business value
  - Avoids technologists as the pilots
  - Aligns business units, partners and suppliers
- Cooperation and Competition is managed
- Company's overall mission, strategy and business remains in control
- Vendor selection is controlled
- Necessary for application, technology and component strategy

# Recommendations

- **Be *selfish*.**
  - Beware of loss of control from external XML standards
  - Create your own Web Services architecture
- **Be *pragmatic*.**
  - Develop style guidelines
  - Create a data dictionary
  - Adopt a service oriented architecture
  - Perform impact analysis on Web Services' effect on bandwidth and CPU utilization
- **Be *opportunistic*.**
  - Seek out the new business value that XML and Web Services enable
  - Beg, borrow or steal what you can.
    - From federal data modeling/XML repository efforts
    - From other standards - internal and external

Set the hype aside and evaluate XML and Web Services for what they are today — a new approach to connecting applications that holds great promise, but contains certain aspects that remain too immature for some current projects. Be wary of using the external form of Web Services and focus on whether, where and when your organization should adopt Web Services as an approach to internal application integration.

Begin planning for the adoption of XML and Web Services by identifying high-value examples that fit the technology-adoption profile of your company. Innovators and early adopters of technology can act today, the trailing minority and majority should begin to consider pilot projects to prove / disprove the concept and its feasibility for adoption with your planning horizon. XML is certainly more proven than Web Services today.

The security, QoS and reliable messaging gaps make widespread exposure of meaningful transactions to the public internet impractical at this time. Focus on how Web Services can simplify the integration problems internally until such time as these issues are resolved — not likely before Q4 2003 at the earliest. Consider whether the document XML approach will solve latency issues.

Educate the organization's business leaders on the current and future capabilities of Web Services. Let educated business need drive which services get the attention of Web Services developers. Draw candidates for "services" from the highest-demand integration projects. Applications that were candidates for Web-to-host tools — the existing functionality is valuable enough to keep, but requires browser or multi-channel, multi-modal access — are likely to be excellent candidates for Web Services. In applications with no discernable API, Web-to-host tools provide a number of options to expose business functions to the Web Services standards, and in that context, enable Web Services to operate in conjunction with custom-built legacy applications.

Service-orientation allows Web services to offer a longer-term vision whereby business functions within and across enterprises interoperate without regard to their technological underpinnings. As such, they will, when mature, provide a significant step forward for integration.