

# zapthink

*research report*

## The "Pros and Cons" of XML



## *The Pros and Cons of XML*

All Contents © 2001 by ZapThink, LLC. All rights reserved. Reproduction of this publication in any form without prior written permission is forbidden. The information contained herein has been obtained from sources believed to be reliable. ZapThink disclaims all warranties as to the accuracy, completeness or adequacy of such information. ZapThink shall have no liability for errors, omissions or inadequacies in the information contained herein or for interpretations thereof. The reader assumes sole responsibility for the selection of these materials to achieve its intended results. The opinions expressed herein are subject to change without notice.

### **Introduction**

A lot has been written, said, and discussed about the Extensible Markup Language (XML). As a result, there is a lot of confusion and chaos regarding exactly what XML means. Is it a language, document, modeling tool, e-Business application, or none of the above? What exactly does XML have to offer? Why does it matter? Furthermore, what does XML bring to the table that other languages, documents, tools, and applications have not been able to do in myriad other ways?

These are very important, if not the most important, questions posed to XML. In the end, the numerous technicalities and intricacies of implementing XML come to naught if there isn't justification for its long-term existence. As a result, this paper serves to comprehensively answer the question "why does XML matter?" It does so not to propose a single explanation for how XML should be used for particular needs, but poses some of the most frequently discussed advantages and drawbacks in using and implementing XML. Hopefully, you will find both justifications as well as challenges in using XML for your needs. The end result is to help make you a more educated reader and help you in understanding what XML has to offer from a viewpoint untainted by hypesters or nay-sayers.

A quick introduction to XML is given in the Appendix of this document for those who are unfamiliar with XML and its technology.

### **Basis for Comparison**

Of course, we can't properly have a discussion of the "pros and cons" of XML unless we are comparing it to something. Since XML has a simple definition, but broad implications, it can be compared to a number of different technologies.

In the e-Business realm, implementers of XML frequently compare it to the efforts of the Electronic Data Interchange (EDI) technology. EDI has been around for at least two decades and is still in major use by diverse industries ranging from automobile manufacturing to health care. EDI is based upon the exchange of "delimited-text" files. The structure of these files is well defined, and is standardized across a broad range of industries. In many instances, EDI relies on private networks known as "value-added networks (VANs)" for the exchange of information, although this is mainly a historical element. EDI requires reliable, secure data transport. VANs provide these qualities of service. VANs also have traditionally other benefits, including the ability to bridge across a multitude of communication methods. Increasingly, EDI has been exchanged over Internet protocols. EDI existed long before "e-commerce" was widely accepted terminology, and can be credited for laying the groundwork for the e-commerce and e-Business application of XML.

However, widespread implementation of EDI has been precluded by a number of major setbacks. In particular, EDI notation is fairly arcane, inflexible, and the reliance on value-added networks (VANs) for their exchange means that implementation and maintenance costs of EDI-based systems are significant. Many cite the cost and complexity of an EDI implementation as the reason why e-Commerce is not widespread among small and medium-sized business. XML offers a number of significant benefits that hopes to solve these problems.

XML is also frequently compared to other structured file formats, such as alternative text and flat-files, proprietary file formats, delimited-text formats, databases, and even SGML itself. XML has some distinct advantages over these formats in specific instances, but there are times when even a comma-delimited text file has advantages over XML. When applicable, this report will compare XML to those more “simplistic” file formats and show when the technology is suited for best use.

Now that we know what we are dealing with, the important questions that remain to be answered are: Why does this matter? How is this better than older or other solutions? How does this make my life easier? This is the thrust of the remainder of this document.

### **A Brief History of XML**

Jon Bosak from Sun Microsystems relates the beginning of XML in the following way:

Like many of my colleagues in industry, I had learned the hard way that nothing substantially less powerful than SGML was going to work over the long run. So from the very earliest days of the World Wide Web Consortium, there was a small group of us who kept saying, "You have to put SGML on the web. HTML just won't work for the kinds of things we've been doing in industry."

Now, the people in charge of the W3C were far from ignorant about SGML. Dan Connolly, in particular, saw very early the need to standardize HTML itself as a proper SGML language, and by the beginning of 1996, he had created a placeholder for some future SGML work within the W3C. But W3C didn't have the resources to pursue this direction, and outside of the few of us who had already been through the development of large-scale electronic publishing systems, no one else really understood the problem.

I had been pestering W3C about SGML and about DSSSL, the SGML stylesheet language, right from the beginning, while I was still working at Novell, and I kept this up after I went to work for Sun. Finally, in early May of 1996, Dan challenged me to put Sun's money where my mouth was -- to organize and lead a W3C working group to put SGML on the web. This was an unprecedented offer, because up until then, all W3C working groups had been organized and run by W3C staff. Dan's willingness to go beyond established practice was the first key development in the process that led to XML.

Dan's offer came just as I was beginning a three-week series of WWW, SGML, and ISO conferences in Europe. This tour put me in touch with just about everyone I needed to talk to about the idea, and by the time I got back home, I had managed to recruit some of the world's leading SGML experts for the "Web SGML" initiative and had secured funding from my management at Solaris Global Engineering and Information Services to carry out the work. This was the second critical turn in the path to XML. Many people know that XML grew out of the expertise of the SGML community, but few people realize even today that the whole two-year effort to develop XML was organized, led, and underwritten by Sun.

It was obvious from the beginning of what was originally called the Web SGML Activity (the name XML was suggested by our technical lead, SGML/DSSSL guru James Clark, several months later) that it would need the support of at least one of the two major vendors of web browsers. In June of 1996 I succeeded in persuading Jean Paoli of Microsoft to join the working group. This turned out to be especially important, because in addition to his SGML expertise, Jean was eventually able to convince Microsoft to adopt the technology.

The basic design of XML was accomplished in eleven weeks of feverish activity under the guidance of editors Tim Bray and C. M. Sperberg-McQueen. The work started in the last few days of August 1996, and ended with the release of the first XML draft at the SGML '96 conference in November. While it took another year to finish working out all the details, virtually every basic feature of XML as we know it today was specified in that first published draft. This remarkable achievement is a tribute to the

team spirit and world-class expertise of the original design group. I am proud to have had the honor of leading this group and proud of my management at Sun for having had the vision to underwrite the effort.

Thus, XML was born – and so too the issues around its use and adoption.

**Point and Counterpoint**

In order to better assist you in determining the positives and negatives of applying XML to solve your particular needs, this report is framed as a series of points in favor or against the use of XML. Each of these points is followed by a counterpoint that rebuts the argument, whether it is a “pro” or “con” argument. Hopefully in doing so, this will help you make the appropriate argument as necessary in justifying or challenging the adoption of XML for a potential implementation.

**Quick Listing of Arguments**

- Pro: XML is a structured text format..... 4
- Pro: XML is designed with the Internet in mind ..... 5
- Pro: XML processing technology is widespread, easily available, and cheap..... 6
- Pro: XML is Human-Readable ..... 7
- Pro: XML is Human-Readable ..... 7
- Pro: XML is very flexible – you can define other languages with XML ..... 8
- Pro: XML separates process from content..... 10
- Pro: XML documents can be validated using a Validating Parser ..... 11
- Pro: XML systems are lower cost than other alternatives. They finally enable small and medium-sized organizations to participate..... 12
- Pro: XML is open. It is influenced by a number of standards organizations..... 14
- Pro: Agreement on a common DTD or Schema results in exchangeable documents..... 16
- Pro: There are an increasing number of individuals with XML skills..... 17
- Pro: Business and technical management positively view XML technologies..... 18
- Pro: XML can leverage existing security solutions..... 19
- Pro: XML is license-free, platform-neutral, and widely supported ..... 21
- Pro: XML can be viewed with simple tools such as browsers ..... 22
- Pro: XML supports complicated and focused searching needs ..... 23
- Pro: XML enables e-Business ..... 24
- Pro: XML is easily internationalized..... 25
- Pro: The Time is Right ..... 26
- Con: XML is a space, processor, and bandwidth hog ..... 28
- Con: XML is just a document syntax, not a programming language or a solution for world peace..... 29
- Con: If XML is so similar to HTML and SGML, why aren’t those languages sufficient? ..... 31
- Con: Every XML format can become a “proprietary” XML format ..... 34
- Con: XML is great for text, but awful for binary data..... 36
- Con: XML is a regression from centralized, efficient database systems to inefficient file processing ..... 37
- Con: XML specifications are not yet complete or tested..... 39
- Con: XML DTDs are insufficient for most business and industrial-strength applications, and XML Schema is too complex ..... 40
- Con: XML will never completely replace other technologies, like EDI..... 43

**Sources**

While it may be impossible to credit particular individuals and sources for each argument, we can say that the end result is an amalgamation and creative interpretation of information gleaned from a few notable and rich sources of XML feedback. This includes the XML-DEV mailing list, XML.org and XML.com websites, discussions on ZDNet, and feedback gleaned from XML-specific mailing lists. Of course, we could not have gotten this far if it wasn’t for the assistance of a few notable individuals who contributed their feedback to help make this document better: Dave Hollander, Ken Sall, Sam Blackburn, Ching-Yang Wang, Robert Miller of GE Exchange Services, and Michael Brennan.

**Pro: XML is a structured text format**

The very nature of XML is that it is a structured document format, in that it represents not only the information to be exchanged, but the metadata encapsulating its meaning, and the structure of the information to be exchanged. Most information has structure of some type. For example, information about a book in turn contains information about the title, author, chapters, body text, and index. In turn, body text contains paragraphs, line text, footnotes, etc. This information is therefore structured in that a document that describes a book needs to describe that information in a way that a person or machine can understand it. Author information should not be contained within the index section, and vice-versa.

Most text files simply cannot offer this clear advantage. They either represent simply the information to be exchanged without metadata, or include metadata in a flat, one-level manner. Common file exchange formats such as comma- or tab-delimited text files merely contain data in predefined locations or delimitations in the file. More complex file formats such as Microsoft Excel contain more structured information, but are machine-readable only and still do not contain the level of structuring present in XML.

Relational and object-oriented databases and formats do present similar advantages as XML, but for the most part, their formats are not text based. Formerly, to get the advantage of a structured format a binary (or machine-readable only) format was employed. Surely, there are other text-based formats that include metadata regarding information and are structured in a hierarchical representation, but they have not caught on in popularity nearly to the extent that XML or even SGML has.

Another advantage of using XML with the use of DTDs or schemas is that XML editors provide structured editing "for free". As a developer, how many times have you run a processor on some formatted file only to get a complaint about a syntax error at line 37? An editor that only allows you to enter valid XML will catch many of these errors as you type them. From another perspective, editors can automatically create a form-style interface from a DTD or schema. Therefore, XML can provide a simpler user interface and eliminate some of the complexity of creating XML documents.

**Counterpoint: Some jobs are still best left to text files**

Despite all the added value in representing data and metadata in a structured manner, some projects simply don't require the complexity that XML introduces. In these cases, simple text files do the job more efficiently. For example, a configuration file that includes a short list of a few commands and their values doesn't require a multi-level, metadata-enhanced file format for its communication. Therefore, one shouldn't take the stance that simply because XML contains structure and metadata it should be used for all file formatting and document exchange needs.

While XML does offer validation technology, it is not currently as sophisticated as many of the EDI syntax checkers. XML editors often lack the detail and helpfulness found in common EDI editors. Many EDI syntax editors can report error details throughout a document, and can complete parsing of the entire document. Many XML editors are unable to proceed beyond the first syntax.

**Pro: XML is designed with the Internet in mind**

One of the main goals in the design of XML was that it be a web-enabled version of SGML. As a result, it works very well over the Internet. XML was created to be a simpler version of SGML that can take advantage of the simple and open transport layers that the Internet provides: email (SMTP), web (HTTP), file transfer (FTP), and other mechanisms. It can even take advantage of some of the native security features such as SSL that are present on the web. Since XML is Internet-friendly, one can make use of any of these transport mechanisms as a means for shuttling XML data back and forth.

Of course, this is in comparison to other formats that have existed much prior to the widespread usage of the Internet. While systems such as EDI have been retrofitted for the Internet, the fact remains that Internet-specific developments, such as namespaces and URI references would be difficult, if not impossible, to implement in older formats such as EDI. With the burgeoning need to communicate in a rapid-fire manner with an increasing number of trading partners, the lack of inherent Internet-readiness would be an impediment.

While text files can also be transmitted over email and the web, structured formats such as relational and object-oriented databases are not easily accessible over the Internet. Their binary-based formats and proprietary connection mechanisms preclude their ability to be easily accessible via the Internet. Many times, gateway software and other mechanisms are needed to access these formats, and when they are made accessible it usually is through one particular transport protocol, such as HTTP. Other means for accessing the data, such as through email and FTP are simply not available.

In addition, since XML can take advantage of existing Internet protocols, designers choosing to use XML in their solutions don't have to create new protocols as a means for transporting the document. Designing a new protocol today may not make sense when existing and well-understood protocols such as HTTP exist. Using these protocols makes the document more portable across multiple platforms, more easily debugged, and easier to understand how to qualify and route. In addition, HTTP as a protocol is well understood, and IT engineers know how to manage the HTTP traffic. Using a new protocol would require inventing some new protocol to go over the wires, which would necessitate identifying new data streams for firewalls, management of the traffic, and a whole ball of wax that is simply not necessary for a structured data format.

**Counterpoint:**

In reality, XML is transport neutral, and many XML applications such as application integration and document management do not even require transport. EDI itself is being successfully transported over multiple transport protocols including VANs, X.400 networks, X.25 packet networks, and even Internet protocols. So, the argument that "XML is Internet, and EDI is not" is not even that valid.

As many EDI supporters will correctly contend, EDI can, and has been, successfully transported over a range of Internet protocols, such as HTTP, SMTP, and FTP. Why then is the Internet such a point in favor for XML then? EDI, after all, is a file format itself, and can be reasonably transported over a range of protocols, including those that have yet to exist.

Also, when users "buy-in" to EDI over a VAN, they are also buying the associated security, authentication, privacy, reliability, and traceability features. Many of these features are lacking in current XML exchanges, or are available from third-party providers that would charge a fee equivalent to usage of an EDI VAN. So, where's the advantage in using XML over the Internet in an EDI-like context? Certainly not security, and if security is available, certainly not cost.

XML inherits the notorious security issues associated with the Internet, but it also inherits the possible solutions to those problems as well. As long as a system is designed with security in mind, exchanging XML over the Internet should be fairly problem-free.

**Pro: XML processing technology is widespread, easily available, and cheap**

One of the primary issues faced by alternate file format and database languages is that processing tools are custom, proprietary, or expensive. When tools are widespread, they are usually specific to the particular file format in question. One of XML's greatest strengths is that processing tools have become relatively widespread and inexpensive, if not free.

Since XML is a structured document that shares many of the processing and parsing requirements as SGML and HTML, plenty of generally available parsers have been built. Many of these parsers are now built-in to general browsers and server-side agents.

In addition, the Document Object Model (DOM) has been created by the W3C as a general model for how parsers and processors should interact and process XML documents for representation as a data-bound tree. As a result, the DOM has produced a generic, universal method for processing XML documents. Applications that require XML processing can access this wealth of tools and specifications and thus add parsing in a relatively pain-free way. A developer does not have to write a new parser, unless they really want to. Many parsers exist in a wide variety of languages, and many of these are free.

**Counterpoint:**

While XML processing tools are becoming inexpensive and ubiquitous, the implementation of those tools is not a free or painless process. Processing XML documents does not stop at parsing an XML file. The data from those documents then needs to be acted on. For most applications, parsing an XML document is just the first step of many. However, standard tools do at least allow programmers not to worry about the step of parsing the document.

In addition, while parsing may be universal, XML tools that reference external resources introduce some processing unpredictability. The main problem is that it is difficult to predict when and even if those resources will be available. Processing self-contained XML documents can usually be done within a predictable period of time, but dependencies often add an unknown amount of time to the work. Thus, some caveats do apply to universal processing of XML documents with tools such as the DOM.

Some business organizations have found the DOM to be impractical for use in their business application environment due to its high execution cost, inefficient error handling, and other implementation problems due to its immature state of development. The DOM is also not the only XML parser or object instantiation technology on the block. As such, standardization may not be as widespread as is believed.

## Pro: XML is Human-Readable

Another oft-cited benefit of XML is its ability to be read and written by humans, rather than created by applications in a machine-only readable format. While many state that XML is primarily used for machine-to-machine communication, and can be created using visual tools that don't necessitate the actually editing of the code, experience with HTML has shown that there are numerous occasions when a developer has to "dip in" to the actual document and make adjustments. It is for this reason that XML is plain text and uses elements that represent actual words or phrases that contain some semantic meaning.

Well-designed XML should be developed with readability in mind. Some programmers would never design and use an XML data structure without looking, sooner or later, at the raw text file. Human readability not only makes debugging and diagnosis easier, but actually speeds up implementation time. With data not locked in a proprietary or binary data format, developers can easily check to see that their processes are resulting in accurate code. The XML format makes this as readable as possible. Otherwise, why bother to make it text in the first place?

### Counterpoint:

As implied in the above argument, if the XML code is intended for machine-only consumption, the human readability argument goes out the window. In this case, since humans may never need to actually touch or look at the XML document, the fact that it is formatted in text and human-readable may actually be a detriment due to the overhead and size of the file itself. Many would argue that XML is bloated for machine-to-machine transfer due to the human readability issue, but this argument will be addressed in full in a later section of this report.

Besides the machine-only nature of some XML code, humans can still create quite unreadable XML code. Many developers code phrases that having meaning for their application but are quite illegible to other humans. For example is the element "<sdjksd23>" meaningful to anyone but the developer and application? What about tags created in a different language such as French or Chinese? Who is the intended reader: developers or business people? Just because XML can be human-readable does not mean that in all cases it will!

## **Pro: XML is very flexible – you can define other languages with XML**

We have already discussed the advantages of the "ML" in XML, but the "X" presents advantages of its own. Extensibility, as applied to XML, is the ability for the language to be used to define specific vocabularies and metadata. Rather than being fixed in describing a particular set of data, XML in conjunction with its DTDs and Schema is able to define any of a number of documents that together form a language of its own.

Indeed, hundreds, if not thousands, of specific document vocabularies have been created to meet needs as varying as healthcare, manufacturing, chess games, user interface design, and petroleum refining. Text files and relational database schema are fairly fixed in that they are meant to represent the information contained within and nothing more. If you need to add a new set of information to a text file or RDBMS... tough luck. XML files, especially those created using an "open content model" can easily be extended by adding additional elements and attributes. Whole classes of documents can be defined simply by sending a document with a new DTD or Schema. Sharing DTD and Schema within a user community results in a joint specification – if not a de-facto or explicit standard.

An argument can be made that so many such vocabularies, initiatives, specifications, and "standards" have never been based on technologies such as EDI, SGML, and HTML. The sheer desire of industries and various communities to define their specific languages around XML is a testament to its ability to accurately represent their needs. If EDI, SGML, ASN.1, and HTML was so flexible and powerful, why aren't a zillion vocabularies defined in their languages?

While XML is fairly simple in nature, in that it only needs to follow basic syntax rules to be considered "well-formed", one of the biggest features of the language is its ability to provide a means for guaranteeing the validity of a document. This means that one can not only send a document to a receiving party, but also send guidelines with which those documents must comply. For example, a guideline may specify that an XML document should contain only the listed set of elements and attributes in a specific order and given quantities.

In the first few years of XML's release, the main means for specifying this validity has been through the Document Type Definition (DTD), a specification based on the format used to specify SGML documents. DTDs use a somewhat arcane and awkward syntax that allows for a simplistic definition of document content guidelines. However, more recently there have been a number of proposals for improved XML validity specification, culminating in W3C's XML Schema recommendation. These have allowed XML documents to have their syntax and schemata defined using XML format, include simple and complex data typing, support for namespaces, and inheritance capabilities.

Even if the XML document is defined and strictly controlled by a DTD or Schema, a user can take advantage of namespaces and Schema to dynamically create documents that borrow from multiple vocabularies and XML-based languages, thus giving the user an amount of flexibility that even standards groups cannot fully reign in.

Thus, XML is not only a human-readable, structured document format that can be exchange over the Internet, but can be a basis for a multitude of other languages, standards, and data formats.

### **Counterpoint:**

As we will address later in this document, the ability to define other languages can potentially be a recipe for disaster, since agreement on a common DTD or Schema is not a guarantee even within a small, well-defined user community. But, more on this later.

In the area of extensibility, XML does have some slight setbacks. In particular, users can't simply add tags to an XML document if it is controlled by a DTD that the user has no permission or capability to modify. In that regard, it is as "locked-in" as a text file or RDBMS format that has a fixed schema. Making changes to

controlled schema is a function of the process used to define the DTD. These processes may be standards-organization controlled, controlled by internal IT staff, or otherwise unavailable for "casual" modification.

Another slight setback to the extensibility claim is that XML files can not be "appended to" as simply as text files and databases. Since structurally, XML requires a matching pair of start and end tags, additions to XML must be inserted in the middle of the document. This makes XML not very applicable to "rolling" file creation needs, such as logging. Although, XML-enabled log files are possible with careful engineering and a certain "flattening" of the XML structure. However, flattening the XML structure really wipes out some of the benefit of the structured nature of the language. Of course, one should always choose the most appropriate language for its intended purpose. As such, XML is clearly not the best solution for all file-formatting needs.

## Pro: XML separates process from content

Since XML represents information and the metadata about that information, it does not specify any particular manner for how the data should be processed or provide any constraints for mechanisms with which to handle the information. This is in contrast to other formats such as text files and databases that explicitly require accessing the documents in a specific manner. Furthermore, the files themselves define how the information is to be processed and what requirements systems must have in order to make sense of the documents. In contrast, XML documents simply encode information and their metadata without specifying how the information is to be processed or displayed.

Often, the capability of XML to separate its process and data content is known as being “future-proof” or “loosely coupled” depending on which end of the marketing and development spectrum you stand. Future-proof in this instance means that no future changes in the data exchange layer should affect the programming layer, and vice-versa. Loosely coupled systems allow for “arms-length” exchange of information where one party does not need to know details of how the other party plans to process the information. These systems are then “loosely coupled” from the existing systems they need to integrate with or whatever system is to be in place in future. This allows for changes in any of the presentation, process, or data layers without affecting the other layers.

This singular feature is a tremendous benefit for those looking for a cross-platform and cross-device language that is meant for simply encoding data and its structure. Of course, the data contained within XML needs to be processed, but the method and means for processing this information is not specified anywhere within the XML document. The Extensible Stylesheets Language (XSL) and its transformation component XSLT specify how XML document can be transformed for display on different devices or transformed to alternate XML representations. This means that a singular XML file can be transformed via multiple XSL files into display on the web, on a cell phone, or into a format for storage and archival. XML documents, devoid of processing instructions, can be exchanged between systems as disparate as Microsoft NT servers and IBM mainframes. Even Internet-enabled toasters that speak XML can communicate with ease with central databases that can understand XML. Thus, XML is a perfect "Enterprise Application Integration (EAI)" language for use in allowing systems to communicate with each other.

### Counterpoint:

It is true that XML offers a "pure" data format that allows users to separate data from its means of being processed and displayed. However, All pure data representations, even common comma-delimited files, can be used as part of a system separating raw data from human presentation.

While XML provides the mechanism to separate data from processing content, in experience, it is up to the human coder to make sure that this separation actually occurs. It is quite trivial to add elements to an XML document that place processing requirements and restrictions on the document, thus preventing cross-platform processing capability. In fact, the potential for creation of "proprietary" XML document formats has many in the open source community concerned. For many programmers, it is unnatural often to abstract the data completely from its processing requirements.

Often misunderstood, EDI does not specify any particular method by which it is to be processed, either. EDI, as a file format, leaves its processing mechanism up to to the business application. As Ajay Sanghi of ABO Software notes, it is possible to send a purchase order in an invoice EDI document! The transaction sets, segments and elements within EDI exist in order to help with commonly used structures but they in no way define how the information is to be processed.

For example, it would be very possible for an operating system vendor to include element "hooks" that are capable of being processed only by machines running that particular operating system or application. While properly developed XML should in theory be platform-neutral, experience has shown that vendors who wish to maintain and protect their platform's market will go to extents to encode elements that are capable of being processed only by their own application suites. The only counter-balance to this natural force is the development of open, cross-industry, widely adopted standards that serve to block the inclusion of application or platform-specific encoding.

**Pro: XML documents can be validated using a Validating Parser**

Another great feature of XML is that documents can be validated for correctness. In most other data formats, errors are not usually detected until something goes wrong or gums up the works. Then, diagnosing the error is usually a tedious process. Imagine having to diagnose a single incorrect element in over 1,000 documents. This is what makes most text files and other data formats a pain in the butt as far as electronic document exchange is concerned.

XML documents, on the other hand, come built-in with error and validity checking. The DTD or schema that is referred to by an XML document can guarantee, at time of document creation, that all the elements are correctly specified, in the correct order. Furthermore, the usage of XML Schema can help guarantee that the values of the element content itself is valid and falls within acceptable ranges. Documents can be validated at time of creation or at time of receipt and be rejected or accepted on an automated basis without human intervention. At design-time, these errors can be fixed before transmission, and at receipt, these errors can be sent back to sender for further human processing with an exact pinpoint as to where the error has occurred.

Validity checking also comes at a very low cost, if not free. Most parsers on the market and available on open source come with validation built-in. While many of these are currently only DTD compliant, the move to XML Schema-based validity checking is well under way. Batches of documents can be checked for compliance against a single DTD or schema, or they can be checked against different schema based on their destination or origination. While use of DTD and Schema does not guarantee 100% validity, it goes a long way in assuring that the vast majority of documents exchanged and received fit an acceptable policy.

**Counterpoint:**

While the use of DTDs and Schema guarantee a certain amount of validity to XML documents, there is no requirement that DTDs or Schema even be used at all. In fact, users can opt to create XML documents that are "almost compliant" and not bother to reference a DTD. Without use of any sort of validation document, there is no way to guarantee that the XML file sent is missing key elements, has out-of-bounds data values, or is sent in an inappropriate data structuring or order.

This lack of validation can be remedied by simply requiring that all XML documents that are sent reference a valid DTD or schema. However, who is to say that those DTDs are the correct ones? XML documents can refer to obsolete, incorrect, or invalid DTDs. While companies may mandate use of a particular DTD, there is no way to enforce that XML documents are required to use a particular DTD or schema.

Furthermore, the difference in validation quality between DTDs and XML Schema is tremendous. While DTDs offer a basic level of checking that an XML document contains accepted elements, attributes, and their ordering, it does not cover data typing, data bounds, inheritance, namespaces, and context-sensitive data values, which XML Schema does cover. Thus, XML documents may choose to use different schema notation with different validation criterion, thus causing confusion at the point of creation or reception.

Therefore, it is not sufficient alone to say that XML documents must use a DTD or Schema, rather enforcement of a particular DTD or schema for the exchange in question is absolutely necessary to guarantee a high level of data quality and validity.

Another big negative comes in the cost that it takes to perform validation on a document. DTD and especially Schema validators are complex beasts that are very slow when compared to other mechanisms for guaranteeing document validity, such as individual RecordSet validation or programmatic post-processing validation. Also, it is less processor-costly to validate at time of retrieval or insertion into a database than to use a validating parser. In many instances, the need to validate at both points in the cycle can create wasted time and added confusion.

In addition, the ability of XML to validate data in and of itself doesn't represent an advantage over EDI, as EDI parsers are also capable of validating data. Most EDI translators validate EDI documents and properly report syntax errors to the sender by way of functional acknowledgment, and in some ways even outperform XML validators in their level of functionality.

**Pro: XML systems are lower cost than other alternatives. They finally enable small and medium-sized organizations to participate.**

Compared to EDI and other electronic commerce and data interchange standards, XML offers serious cost and development time savings that make implementation of XML good for the bottom-line. There are many components to document exchange and electronic commerce systems: document creation tools, processing components, validity checking, data mapping, back-end integration, access to a communications backbone, security, and other pieces of the commerce puzzle. XML greatly simplifies, if not eliminates, many of these steps.

XML's built-in validity checking, low cost parsers and processing tools, XSL-based mapping, and use of the Internet keep much of the e-commerce chain low cost. In many cases, general XML tools can be found that are not only applicable to the solution to be solved, but are flexible and very inexpensive. Whereas EDI is a specific domain of knowledge and expertise that comes with a comparable price tag, XML makes use of technology that has been in use in the open community for years, if not decades. Systems that take advantage of this wealth of available processing power and know-how will greatly reduce not only their cost but also their time to implementation.

The use of the Internet itself lowers greatly the barrier for small and medium sized companies that found EDI too costly to implement. Simple functionality, low-cost tools will go a long way in helping these companies afford to exchange high-quality, structured documents that are capable of supporting commercial exchange and back-end integration.

**Counterpoint:**

While many of the costs for processing and exchanging documents are eliminated by the use of XML, not all costs are eliminated and some are actually increased. First, it is important to realize that systems must be enabled to communicate in XML before its benefits can be realized. How many systems are involved in the commerce process? How many of these systems must be able to understand XML? What is the cost in modifying these systems?

Regardless of the end data format, most B2B exchanges must include workflow error handling, data mapping tools, the communication backbone, and partner management. These functions aren't features of EDI -- they are the requirements for B2B. Now, it should be noted that mapping and the communications backbone may be solved in some minds by the use of HTTP and XSL, but the complexities of mapping and the needs of the communications infrastructure should be evaluated. Does HTTP offer a secure and reliable enough transport for XML messages of a B2B nature? Does XSL perform the appropriate business-context transformations required? Companies such as WebMethods are offering XML-based solutions to these problems, but the solutions are not cheap. As EDI has taught us, not only the "big boys" need to be able to play in this space. Small and medium size companies have the same B2B integration challenges with XML that existed with EDI.

Many will argue that modification of "legacy" systems is not actually necessary, since gateways and translators can be built that transform data from XML to the native communication format. However, this does not eliminate entirely the cost of XML integration, but shifts it from the system to be integrated to the gateway or integration service. Gateway systems, by their very nature, will be custom implementations and will require the time of a developer, additional software costs, or both. Thus, integrating with back-end systems is not a "free" endeavor by any means, but is mitigated by the costs involved in getting non-XML systems to be suddenly XML-aware.

In addition, XML cost comparisons are frequently made to EDI, which is indeed an expensive solution for small and medium-sized businesses. However, the cost comparisons against simple flat-file or proprietary file formats are not as clear. Many of these same businesses participate in the e-commerce process by exchanging simple files that are both easy to create and process. The workload and cost is merely

distributed to the receiver of the document, rather than the creator. In these cases, usage of XML may in fact result in higher cost of implementation for the small and medium-sized businesses.

In order for cost to be used as the primary justification for usage of XML, a careful consideration should be taken into account for how systems will consume and output XML information, and what impact this will have on the receivers of this information. Don't forget to take into account the cost of customization and integration, even for gateway systems!

**Pro: XML is open. It is influenced by a number of standards organizations**

Whereas certain types of flat files and database formats are proprietary and their development is under control of a single company, XML is a product of an open standards process and continues to be developed as an open source, community effort. Furthermore, many vocabularies and languages that are derived from XML are also being developed with open source processes. This means that developments are subject to processes that are beneficial to the users of XML. Rather than being "beholden" to the error-prone, accelerated release schedules of a commercial organization, the core XML standards are approved and released as part of a standards organization process. While this may some times result in longer-than-usual development times, the end result is that organizations that support XML are not "placing their bets" on a specific company or technology.

Of course, all this discussion of "open" decries an accurate description of what precisely that mean. The W3C's implied position is that when they describe XML as "open", they mean that it isn't a proprietary tool. In that sense XML, including its specification and its associated technologies are open. But the openness of XML doesn't extend to the data formats or information systems that XML interacts with or to the information that it encodes, each of which can be as proprietary and "closed".

However, we are describing here not only a specification that is "out in the open" and can be viewed in its entirety by all interested parties, but also an "open process". An open process means that the forces and efforts that are employed in the creation of the specification itself are open. Meetings are publicized, held outside the confines of a single dominant company, and voting processes for modifications to the specification are well understood. Most importantly, any party that is interested in contributing and can bring resources to bear on the problem should be allowed to contribute in a truly "open process". While W3C and other organizations follow this open-process model, not all other XML specifications and "standards" do.

Another good definition of "openness" comes not in the definition of the specification, but in the manner with which it is used. XML's openness means that it can be created by Corporation A's tools and processed by Corporation B, C, or D's tools or open-source applications and tools, or be created and processed in any combination of different tools and applications by different or competing tools vendors.

For vendors of software applications who use "open" XML protocols and standards, this means that their software can be replaced. This is primarily an advantage to the consumer who has increased choice in who and how they choose to have their problems addressed. But this is also an advantage for the software vendor in that they can develop open interfaces that keep their software applications always current and open for modification. In addition, no company can do everything well. The great server vendors may be mediocre players in the middleware, desktop, and editor environments. The adoption of open standards allow them to "play well" with others in the space and reinforce their own products' best features.

**Counterpoint:**

"Open" is a four-letter word. The number of ways in which the term is misused far outnumber the instances in which it is appropriately applied. Truly open standards and processes are few and far between. This is primarily due to the fact that open processes result in standards development times that are exceedingly drawn out due to the consensus nature that open processes demand. It is far easier to develop an XML specification in short time when an individual organization or small, a controlled group is behind the steering wheel. Of course, then the application of "open" is not really applicable, no matter to what extent the work is publicized.

Some of this "openness" debate has even extended to the W3C, a supposed model for open standards processes. For example, some would argue that the development of XML Schema was a somewhat closed process in which the interests of certain large organizations and the pressure for a timely release unduly influenced the specification outcome. While this may not actually be the case (separating truth from

perception becomes difficult in situations like these), the result is that the credibility of an organization in maintaining their "open" outlook becomes somewhat tarnished.

Therefore it behooves all organizations looking to adopt a standard of any sort to investigate the processes by which it was created, its level of completion, its ongoing development process, and how it applies to your specific needs. If it seems that a particular specification is subject to the undue pressure of certain organizations that are not favorable to the specific development needs of the company, then the standard itself may not be worth supporting.

Of course, all this talk of "openness" really has to come down to a bottom-line. If a standard doesn't really need to be open in order for it support one's needs, then who cares who is developing the specification and how it is delivered?

Some would also argue that regardless of the amount to which a standard is "open", there are specific problems that face standards development bodies. Many would say that XML specifications may be reliving the mistakes of previous standards efforts.

The first common mistake, as we mentioned earlier, is the ability for a standards organization to maintain true vendor-neutrality. Prior standards working group experience has shown that vendors will serve on standards bodies and produce work only if it is aligned with their competitive needs. As a result, standards organizations have to align with the strategic imperatives of their constituent working group companies if they hope to see their specification adopted in any meaningful way.

Another common mistake is that standardization efforts see themselves as products that need to be "pitched" and "marketed". These standards organizations then pitch their efforts as the latest in a series of great promises for solving the major problems present in their space. These promises have been voiced for technologies such as Unix, the object component protocols, Java, and various network protocols. Over promising and under-delivering on these promises is a recipe for sure failure. In addition, delivering half-baked and under-specified standards releases is sure to kill any marketing or favorable momentum generated by publicity efforts.

Another common problem is that many standards efforts attempt to extend their reach into areas that are not applicable. For example, at what point does it make sense for a biology standard to apply to other fields such as insurance? Just like any successful business, a successful standard is one that delivers a valid value proposition to a focused user community. Dilute the value proposition or the user community, and the standard is sure to fail.

Furthermore, the work that is done by standards committees falls into one of two camps, for the most part: the least common denominator (LCD) or the greatest common denominator (GCD). As a result of the constant tug-of-war present in standards working groups, final specifications are a compromise of one of two sorts. In LCD specifications, the specification reflects all the elements that could be agreed upon by all parties. If this means that a specification with 60 elements was whittled down to only 10, then the result is the lowest common denominator with which all parties can agree. Another variation on this is that the specification contains all the suggestions of all the parties. This means that everyone at least has some of his or her suggestions embodied in the final result. This greatest common denominator approach results in a fat, bloated specification that is too large for everyone and not specific for anyone. LCD approaches result in specifications that are customized with add-ons that are often proprietary and company specific. GCD approaches result in specifications that are partially implemented on a selective basis with companies at odds over which parts of the specification they will choose to implement. Either solution is a poor choice for the implementing company.

Finally, many proposed standards overlap in their specification, causing confusion. In supporting one standard, a company may be inadvertently closing the doors on accepting another, conflicting standard. Until the proverbial "standards shake-out" occurs, a company is forced to make choices between standards that may still be immature and in development.

## **Pro: Agreement on a common DTD or Schema results in exchangeable documents**

Implementing a standard in XML is quite simple – all that is really needed is shared DTDs or Schema. In essence a standard is just an agreed-upon set of elements, attributes, structure, semantics, processes, and vocabulary with which documents can be exchanged. All of these, with the exception of processes, can be easily represented in DTDs and Schema. Even business processes are being represented in a series of documents which can be shared among a user community.

By agreeing on these documents, a standard can be born. Of course, adherence to the standard is the primary concern. When individuals or organizations begin to add their own elements, use different vocabulary, or use the schema "not as intended", things begin to break down. Whether or not in practice standards are appropriately adhered to, XML provides adequate technology for specifying standards and assuring that documents prepared are valid.

Well defined standards have the notion of extensibility built-in. Such standards allow for later definition of additional information fields, expansion by implementing organizations, and other "customization" needs. This is usually accomplished by providing areas within the standard for "plug-ins", DTD stubbing, and use of "recursive trees" that allow the specification to be expanded.

### **Counterpoint:**

EDI has shown that even if a specification is as complete as possible, individual implementations can vary tremendously. As such, additional agreements and specifications are needed between companies that are using a standard. These "guidelines" are constantly changing and reflect the needs of the companies to appropriately reflect their usage of the data interchange technology.

Even when DTDs are not modified, their vocabularies and semantics are many times misunderstood. For example, an element called "AA" in one XML document referencing a particular DTD may have a very different interpretation or definition from the same "AA" element in another XML document that references a different DTD. In one case, the "AA" element may refer to an offer for sale, while the other may refer to an offer for purchase. Imagine the confusion that would result if the sending system interpreted the DTD as an offer for sale, while the receiving system thought it was an offer to buy and sent an invoice!

Versioning and compatibility issues are also a primary concern with shared DTDs. If a DTD is "proprietary" in nature, created by a single corporation or organization, then all groups making use of the DTD have the responsibility of keeping up with changes in the format as they occur. This is especially the case if a DTD is linked to a product. The company has the ability to change file DTDs every release, and users will have the challenge of keeping up with an ever-changing format.

In order for XML to become a widespread success, companies and organizations of all sorts will need to adopt international standards in a consistent manner. Buyers should insist on products that adhere to these standards. Support of XML itself is not enough; users will be sorely disappointed when they discover proprietary DTDs getting in the way of system integration.

**Pro: There are an increasing number of individuals with XML skills.**

Due to XML's popularity, ease of use, and increasing proliferation of tools, the number of individuals and organizations skilled in XML use is increasing exponentially. It is becoming considerably easier to find skilled employees and contractors that are familiar with XML, the standards, and best practices for implementing XML in multiple environments.

This proliferation of trained resources is a direct result of XML's open and standard nature. XML training is becoming standardized, since companies can hire individuals who are trained in the open standards versus familiarity with specific products or versions of those products. The number of training resources such as books, online training, in-person training, developer web sites, conferences, seminars, and additional means of gaining information on XML is also increasing. With such a quantity of people walking around with XML in their heads, there is no doubt that XML will continue to have a bright future.

**Counterpoint:**

While getting employees and resources that are XML familiar may be increasingly easier, their familiarity with common business processes and the pitfalls that e-Commerce and e-Business efforts entail may be lacking. A side effect of the rapid growth of the Internet generation of practitioners is that many are self-taught, and have primarily HTML and Java programming skills to the exception of others. Most of these individuals usually have not had formal exposure to data fundamentals in general and the relational model in particular. The rapid emergence of "green" XML developers may in essence be a setback for some that may depend on developers with the knowledge to not repeat errors made years earlier in other efforts, such as EDI. It may not always be sufficient to find "XML experts", but to also find "business process experts" that are familiar with XML. Hopefully, these too will be easier to find in the near future.

**Pro: Business and technical management positively view XML technologies.**

As one XML user states, "XML is hip, happening, now." EDI is perceived as crusty and old. Text files are blasé, and databases have increasingly become a staple. The idea that XML represents a new, fresh approach to solving many lingering problems in a flexible manner appeals to many in senior management. In many instances, buying into a new technology requires the approval of the senior levels of IT, if not corporate, management. With XML's continuing positive exposure, getting buy-in on an XML project is become an increasingly simpler endeavor.

Market awareness should never be underestimated when it comes to technology adoption. Some technologies, such as Expert Systems, that have had years of development and significant technology improvements have lacked long-term sustainability due to the lack of buy-in by senior members of management. Of course, this has to do with many of the other points in this document. Buy-in comes from a justifiable business case that validates how, when, and why a particular technology should be used.

**Counterpoint:**

Hype is never a good reason to select a technology. If the business benefits, technological advantages, resources, and competitive advantages are not present in a given technology, it should not be adopted, plain and simple, regardless of what others are saying. Remember what your parents told you about peer pressure.

**Pro: XML can leverage existing security solutions.**

Since XML can make use of the existing Internet and network infrastructure, it can take advantage of the increasing framework for providing different degrees of security. When one says "security", one is actually speaking of a body incorporating many different concepts:

- Encryption ("wire-level security") – protecting data from prying eyes
- Authentication – making sure the receiver of data is who they say they are
- Authorization – even though you may be who you say you are, do you have permission to access the data?
- Privacy – you can access this data, but nobody else.
- Permissions and Data integrity – don't mess with the data

In each of these areas, XML offers credible, existing solutions that have been tested with other technologies. In the area of encryption, XML can be transmitted over encrypted wire-level protocols such as Secure Sockets Layer (SSL), the protocol used in HTTPS for securing web sites. Authentication is becoming increasingly enabled using Digital Certificates and Digital Signatures. Authorization techniques and methodologies work just as well with XML as they do for other documents and protocols. Privacy is becoming a key issue in the past few years, and the W3C is doing its part to address this concern by releasing the Platform for Privacy Preferences (P3P). XML's use of DTDs and Schema can even address the issue of preventing unauthorized data manipulation, although XML is a tad weak in this area.

**Counterpoint:**

XML introduces new security risks of its own that could pose serious threat to the integrity and exchange of data. In particular, the external referencing of DTDs poses some security concerns. Authoritative DTDs for such standards as XHTML will most likely be posted on numerous sites for external referencing. Due to their significant stature, most developers will use and trust those references, without being aware of the security of those web repositories.

Some of the attacks on the security at those sites can take the form of "messing with the code" in order to break validation. Simply adding extraneous declarations to a list of character entities can do this. Any change that produces a fatal error in a DTD can then produce a chain reaction damaging XML processing on a large scale. Other manipulations, such as changing attributes from being optional to required, are possible and can be difficult to track down. Also, a malicious intruder can redefine default values for attribute, thus causing havoc and potentially exposing referencing sites to security holes and further attack.

Intruders can also manipulate entities so that they insert text into documents. Because XML 1.0 permits multiple entity declarations, and the first declaration takes precedence, it's possible to insert malicious content where an entity is used. David Megginson once demonstrated this by inserting the Communist Manifesto in every occurrence of the "&dash;" entity.

The only counter to these threats is the securing of referenced sites, or the copying of references to secure local machines so that they are no longer vulnerable to such distributed and high-impact attacks. Of course, doing so removes many of the benefits that external referencing provide. Another solution is to use a syndication model for retrieving new versions of DTDs and Schema or to use Digital Certificates and Signatures to authenticate the source of creation.

As XML spreads to back-office systems, databases, and applications, and is used on an increasing number of Web sites, the powerful capabilities of XML's ability to seamlessly link data sets will create a new slew of security problems. The code defined by XML tags can carry virtually any payload through the firewall unchecked. Malicious XML code could get a free ride into almost any organization because firewalls and filters trust that the XML tags are honest in what they describe. While XML developers state truthfully that XML is just a markup used to convey information and build applications, the manner in which it is used to convey information and build applications will create security concerns.

XML is also vulnerable to text-based attacks that formerly could be thwarted by filtering schemes. Text-based attacks are accomplished by inserting complicated text information, such as character, symbols, and numbers, in arbitrary locations in documents and applications. While text-based attacks were successfully filtered by firewalls in the past, XML introduces a new twist in the form of Unicode, which can represent text information in almost infinite ways. Programs that block bad code won't work with Unicode due to all the ways the information could be written. Also, firewalls can't check XML-embedded data due to the complexity and processing needed to do so. Since XML is a data format, there will be no evidence of intrusion until the effects of the bad data is felt in an application.

**Pro: XML is license-free, platform-neutral, and widely supported**

As mentioned earlier, XML is a technology that has no single owner or point of commercial licensing. As such, it can be freely implemented in any application or usage scenario that an organization sees fit without incurring licensing costs. Due to its separation of process from content, it is also a good example of a platform-neutral data format. See earlier discussion on the separation of process from content for a more thorough discussion of how this benefits the ability to exchange data on disparate platforms.

In addition, XML is widely supported by individuals and organizations of all sorts. As a truly open-source and open-process technology, XML provides implementers a wide base of resources that can provide assistance in its implementation. Rather than being constricted to getting technical support and assistance from a single company, XML provides implementers the opportunity to obtain a XML-based product from one company, implementation services from a second company, and support and ongoing maintenance from yet another company. This is the essence of the open-source movement.

**Counterpoint:**

Even though XML is in itself a license-free, platform-neutral, and widely supported technology base, applications and other technologies built using XML are not guaranteed to contain similar qualities. Many companies will be apt to make use of XML to create commercial, proprietary, and less-supported products. Companies should beware claims that because XML is open, so too will be all applications that are built with it.

XML can be likened to ASCII text files. ASCII text files are also license-free, platform-neutral, and widely supported file formats, but many companies have developed text-based file formats that are proprietary in nature. It's not XML, but how uses it and licenses it that counts.

**Pro: XML can be viewed with simple tools such as browsers**

In the same vein as the argument regarding easily available and inexpensive XML processing tools, there also exist a number of low cost methods for visualizing XML.

These methods fall into one of three camps:

- Internet browsers natively viewing XML, or using XSL or CSS to render on the browser
- Conversion of XML to HTML at server-side, run-time or in batch using XSL or other methods.
- Use of specialized Java applications to render in browser

The reasons for the visual display of XML documents are plentiful. There are many instances when viewing an XML document is necessary in one manner or another. These instances include the need to edit XML files, visual creation of XML documents, debugging, and human processing of XML files that "fall through the cracks" of automated systems.

In addition, there are an increasing number of visual tools for the creation, editing, debugging, and manipulation of XML documents are emerging. Many are low cost, if not free, and are very well suited to the tasks necessary in manipulating XML files. These tools not only allow a user to gain access to the "raw" XML document, but to gain a visual interpretation of the metadata in a graphical manner, allowing for navigation within XML files of a substantial size.

It is important not only to process and otherwise make use of XML documents, but also to make it easy to create, manipulate, and debug those very same documents. Fortunately, XML has provided a rich technology base from which doing so is low cost and sophisticated.

**Counterpoint:**

While it is possible to use browsers as a freely available XML visualization tool, browser support for XML is currently spotty and not universal. However, the most appropriate question is who needs to view the XML data and why? If the XML information is meant to be machine produced and consumed, then the likelihood of a developer needing to poke their way into what may be unrecognizable XML code is low. More necessary are business applications that understand XML and can interpret the documents in a manner that is relevant to the context in which it was produced. After all, if a business user is entering a purchase order in a system, they will be unable to debug an XML document if the other party rejects it. Instead, the tool used to create the document will have to interpret the results and present them back to the user in a consumable manner.

So far, these business and context-sensitive tools have yet to appear. In general, the usability of XML is currently geared towards developers and programmers. Another evolution in XML is needed before the business-level public can generally consume it.

## **Pro: XML supports complicated and focused searching needs**

One of the things that metadata enables is the encoding of data with extra information that aids in its archival and search. As such, documents that formerly were just bunches of text are now documents rich with metadata that is ripe for searching and proper archival.

This aspect of XML is greatly assisting its widespread use and adoption by industries that have volumes of data waiting to be mined. It also is revolutionizing the web search engine industry. How many times have you entered a term in a web search engine only to get tons of irrelevant responses in return? For example a search for the term "bulb" would return items concerning light bulbs, flower bulbs, and dim-bulbs. On the other hand, if we could search on a specific tag for the value "bulb", we would get only the documents we are interested in. This notion of metadata-assisted searching is melding with former concepts in the knowledge management and artificial intelligence communities to result in a "Semantic Web" that not only encodes data but its semantic meaning.

In a more practical and immediately realizable manner, however, one can simply use the metadata as part of the keywords in searching efforts. For example, a librarian can search for all literature with "Koontz as the author, published in 1995 or later" rather than doing a series of text searches. Many search engines use such inaccurate concepts as relevancy, word occurrence, and word proximity for accurate searching. However, this is made obsolete if users can pinpoint their searches by refining them using metadata contained within documents. XML enabled this sort of search and archival as a side effect of its structured text format.

The ability to limit search results to documents within a particular tag set is one of the market forces that will help foster usage and adoption of XML.

### **Counterpoint:**

The primary challenges in making use of metadata-enhanced searches and archival are in encoding the content and enabling servers to use that metadata. Currently, the tools to accomplish these goals are not prevalent for these kinds of activities. Tools need to be created that make it easier for people to add metadata to existing documents or create documents from scratch with the necessary metadata that allows for later searching. Search engines need to become XML-aware and be built with an understanding for how metadata should be processed and archived. Currently, the lack of a robust set of metadata enhancement and search tools makes the concept of using XML as a refined search tool a concept that has yet to be delivered.

The promises of the Semantic Web may yield some solutions to these challenges, but it too is just promise and has yet to be delivered as a robust set of tools and technology.

**Pro: XML enables e-Business**

The nearly simultaneous emergence of XML and Internet-enabled e-Business has resulted in a wealth of offerings that combine these two major technologies. XML-enabled e-Business is rapidly becoming a reality. Standards, tools, applications, and services have emerged that have leveraged the power and simplicity of XML into creating flexible, robust, and powerful e-Business systems. As a result, many of the strengths that have formerly been realized only by implementing EDI can now be put into place by using a suite of XML-based offerings. The "stack" of XML standards supporting these e-Business offerings is improving on a daily basis. Currently, a user has a plethora of standards and specifications to choose from: RosettaNet, ebXML, UDDI, cXML, etc.

The reason for this growth is due to the ability of XML to solve some of the major problems associated with managing business-to-business commerce. One of the challenges in conducting e-business is communicating with other organizations, whether they are partners, suppliers, competitors, or even other groups within the same company. The use of accepted XML standards simplify business-to-business communication because the only thing that any two organizations have to agree on is the implementation of those standards. Organizations don't have to know the details of each other's back-end systems. This means they can maintain complete neutrality as to operating system, database type, programming language, and even to a certain extent their internal business processes.

Once a common standard is adopted, internal systems can be mapped to these standards and back-end systems can then be engaged in the "business of e-Business". One of the primary benefits of this neutrality is that additional trading partners can be added to scope of a business' trading network without requiring recoding, additional mapping, or further development to be able to include that partner in the trading process – as long as the partner adheres to the same standards.

**Counterpoint:**

XML wasn't the first to implement e-Business in a credible manner. Remember EDI? The concept was much the same. Trading partners would all agree to use the same standards, albeit not as structured and using a private network, and the companies would be able to seamlessly perform business-to-business transactions. Well, this didn't work as planned for a simple reason – businesses will need to modify the standards to meet their particular trading needs. This then breaks the supposed claim that additional work doesn't need to be done to add trading partners to the network.

In addition, e-Business applications require shared agreements about data exchange above and beyond the simple document interchange level. These higher layers include service discovery, business process management, and messaging standards. Standards in this arena need to develop to the point that they are pervasive and flexible. Flexibility needs to be built-in to the standard so that small deviations don't break the overall transaction integrity.

**Pro: XML is easily internationalized.**

One of the drawbacks to EDI and some text file and database formats is that they don't easily support the needs for internationalization and localization. Currently, in those languages it is difficult to represent information contained in a Unicode alphabet. XML as part of its initial specification supports these needs inherently.

XML syntax allows for international characters that follow the Unicode standard to be included as content in any XML element. These can then be marked up and included in any XML-based exchange. The use of internationalization features helps to surpass one of the early problems of other formats that cause unnecessary schism and conflict between different geographies. For example, it is not fair that an English technical manual can be marked up in a file format if a Japanese manual can't be likewise formatted. XML sought to solve this problem from the get-go.

Of course, just because XML can be internationalized, it doesn't mean it will. It is up to the programmer to make use of XML's features and properly encode the data. Developers should understand how to take care of handling character encoding properly in order for XML to be properly internationalized.

**Counterpoint:**

While XML gives the freedom to use any Unicode character within the text of an XML document (as content for elements), it strictly limits the characters that can be used as names for XML elements. With the addition of new characters and support for new language scripts in Unicode 3.1, these naming limitations have become increasingly restrictive. This means that XML documents are limited to using a subset of Unicode characters as names even though they can use any character within element content.

However, the XML Blueberry project aims to solve this small "glitch". While many would argue that the need for native-language XML elements and attributes is overkill and unnecessary (after all, we're not talking about XML document content, but just the names for elements), the addition of these new features aims to patch this loophole in internationalization support.

To further clarify this fact, Martin J. Duerst, Internationalization Activity Lead for the World Wide Web Consortium has informed me:

A very wide range of characters from all kinds of scripts (Latin, Greek, Cyrillic, Hebrew, Arabic, Armenian, Georgian, 8 Indic scripts, Thai, Lao, Korean Hangul, Chinese/Japanese/Korea ideographs, panese Hiragana/Katakana, can be used for element and attribute names. See <http://www.w3.org/TR/REC-xml#CharClasses> for the full list. This list is based on Unicode 2.0. In the meantime, mainly with Unicode 3.0 and 3.1, new characters have been added. These include scripts such as Ethiopian, Mongolian, Cherokee, Khmer, and Runic.

One part of the XML Blueberry Requirements (<http://www.w3.org/TR/xml-blueberry-req#requirements>) is indeed about extending the list of characters for element and attribute names from Unicode 2.0 to Unicode 3.1. It is definitely not fair if some languages (those covered by the characters in Unicode 2.0) can be used in element/attribute names, and others (those only covered by Unicode 3.1) cannot, and this ultimately has to be fixed.

The addition of these features will be used only by those that are interested in these particular internationalization features, and theoretically shouldn't break older parsers. It should be trivial to upgrade parsers by expanding a few tables. However, this remains to be seen.

## Pro: The Time is Right

One of the simplest arguments in favor of XML is that the timing is right for adoption of technology such as XML. Even if XML doesn't offer any significant technological advantages over other offerings such as EDI, SGML, ASN.1, or HTML, it comes at a time when the environment is ripe to adopt such technology. For one thing, never before, or at least not since ASCII, has any one data format been so accepted by such a diverse, and often competitive, group of user communities. Since when have Microsoft, IBM, Sun, Oracle, HP, Novell, and countless others all agreed on the same representation for data? This common agreement on XML as a means to represent structured data is unprecedented. EDI, SGML, ASN.1, and even HTML have not garnered the same amount of overwhelming, coherent support.

In addition to this overwhelming support, XML requires a certain amount of bandwidth, storage, and processing power to work effectively. Twenty years ago, it would not have been feasible to propose XML in its current form because most machines had less than 1 Megabyte of memory, processing speeds well under 50 Megahertz, and storage capacities seldom exceeding 40 Megabytes. This may have been part of the issue in processing and working with the complexities of SGML.

Another aspect of timing is the fact that the Internet exists. Without the Internet, it is arguable that XML would never have been proposed. The ubiquity of a communications platform, and our standardization using protocols such as HTTP, FTP, and SMTP allows us to make certain assumptions about how data is transferred that we simply would not have been able to make a decade or more ago. Back then, we had 1200 baud modems connecting to proprietary, dial-in systems. The ability for mass-exchange of data would have been almost impossible on any large scale. The presence of the Internet has contributed to an environment where the timing is right for the dissemination of XML as a data format.

A more subtle aspect of timing relates to the general atmosphere for "open systems". A decade ago, it was not in vogue for companies to develop software, systems, and standards that would be placed in the public domain or submitted for "open source" collaboration. The hegemony of certain software and hardware companies contributed to an atmosphere where technologies and standards were guarded aspects of platform dominance. With the emergence of a number of major open standards and open source efforts including Java, Linux, and even HTML, and the proliferation of technologies such as Peer-to-peer networking, the community is more likely to accept an open technology such as XML. A decade ago, it would have been inconceivable to think that Microsoft, IBM, or even Sun would contribute their energies to the development of technologies and standards that would then be shared with their competitors in an open manner.

Twenty years ago, it would not have been feasible for any of these conditions to have been met, even if XML was proposed in its current form. So, in response to SGML, EDI, and ASN.1 enthusiasts that claim that XML offers nothing new, the answer is: maybe they are right, but those technologies were simply proposed at the wrong time. Many technologies succeed and linger not because they embody revolutionary change, but because they represent the readiness of the community to accept them.

### Counterpoint:

Even if XML was a technology whose timing was right for adoption, it is simply too early to tell if it has any longevity. Despite the overwhelming show of support for XML by an unprecedented number of competing vendors, XML may merely be the latest "fad technology" in the industry. Many other "holy grail" technologies have been proposed, and failed, simply because the initial outpouring of support never resulted in widespread adoption and return on investment. The end result of any technology is that it needs to create a lasting and compelling value to the implementers that use it. To use XML simply because its timing is right is not a good enough argument for most business-types. Who cares if everyone and their mother is behind XML if it doesn't have a lasting, compelling value?

As for the ubiquity of the communications platform, it can be argued that those who were using EDI already had a ubiquitous connections platform in the form of the EDI VAN. Even though efforts have resulted in a means for communicating EDI over the Internet, there was no need to do so a decade or more

ago. The presence of a robust, secure, and reliable transport mechanism for EDI made it just ubiquitous as a means for exchanging business data as the Internet.

Also, while IBM, Microsoft, Sun, and others are currently agreeing on their implementations of XML, the pressures on the usage of XML are tremendous. It is hard to say how XML may splinter into different forms among competing vendors. Perhaps SOAP, ebXML, or other XML-based initiatives may deviate into a number of incompatible implementations. Or maybe the very features of XML itself or its core components such as XML Schema may be altered so that they are only consumable on certain platforms. Surely, while today's support of XML is overwhelming and "it's time has come", it is hard to predict how this environment may change.

## Con: XML is a space, processor, and bandwidth hog

One of the most notable and significant "knocks" against XML is that it's huge. XML takes up lots of space to represent data that could be similarly modeled using a binary-format or a simpler text file format. The reason for this is simple: it's the price we pay for human-readable, platform-neutral, process-separated, metadata-enhanced, structured, validated code.

And this space difference is not insignificant. XML documents can be from 3 to 20 times as large as a comparable binary or alternate text file representation. The effects of this space should not be underestimated. It's possible that 1 Gigabyte of database information can result in over 20 Gigabytes of XML encoded information. This information then needs to get stored and transmitted over the network – facts that should make Intel, Seagate, and Cisco very happy indeed!

Let's not also forget that computers need to process this information. Large XML documents may need to be loaded into memory before processing, and some XML documents can be Gigabytes in size! This can result in sluggish processing, unnecessary reparsing of documents, and otherwise heavy system loads. In addition, much of the "stack" of protocols require fairly heavy processing to make them work as intended. For example, SOAP, which is a cross-platform messaging and communication platform for use in Remote Procedure Calls (RPC), both between and within server systems. The marshalling that occurs in between can cause system performance to be quite poor since XML is after all a text-based protocol that is being used to make RPCs between systems. Using XML in this transactional, real-time manner may impose more requirements on the system as far as parsing and processing than the system can handle.

In addition, a problem of many current XML parsers is that they read the entire XML document into memory before processing. This practice can be disastrous for XML documents of very large size. XML is not only a data language, but a complicated one at that (from a parsing perspective). It often times increases code complexity because XML can be more difficult to parse than a simpler data format such as comma or tab-delimited fields.

### Counterpoint:

Many of these problems can be addressed in a number of ways. The first is to not make use of XML everywhere! In the places where it is used, one should evaluate the way it's being used. Is the XML being used as a storage or messaging format? Does it need to be used real-time, or is it asynchronous or batch in mode? Solutions can then be addressed for each of these points.

For those using XML in a batch or non-real time mode, let's be real: disk and bandwidth costs are getting cheaper every day. There is no doubt that XML would never have worked 20 years ago. The only reason we are even considering it today is because it costs almost nothing to store and exchange this data. This will get even more insignificant as disk and bandwidth space comes down to pennies per gigabyte.

In those situations where real-time processing and processing power are primary concerns, there are a set of technologies that compress the XML into a binary format for transmission across the wire, and are then decompressed before processing. The HTTP specification supports transport-level compression using standard compression formats such as gzip, and although few people implement it, compression rates of up to 90% can be achieved using gzip on XML data. For those looking to transport XML across other protocols, there are other similar technologies that can offer similar compression ratios. After all, XML consists of many angle brackets and white-space characters!

Compression addresses the bandwidth transfer issues, but not processing requirements. These issues are addressed by technologies that only load a portion of the XML document into memory and then selectively parse it for information needed. Thus, a 1GB XML document can be simplified to the minimal elements necessary for a particular transaction.

As with any technology there are certainly drawbacks, and size is the primary issue for XML. But, there is no reason to discount a technology merely because it imposes more requirements on disk space, bandwidth, and processor power.

## Con: XML is just a document syntax, not a programming language or a solution for world peace.

Someone once said that XML is better than sliced bread – it doesn't mold. With all this XML hubbub, people tend to forget that XML isn't an application. It's not a programming language. It's not the answer to world peace and starvation. It's not even a breakfast cereal. XML is simply a document format that has characteristics that make it very well suited to sending structured information containing metadata that is easily validated.

This misalignment in understanding causes quite a bit of confusion not only about what XML is, but what it is capable of doing. XML require parsers and applications to process them. Without them, the format is as useful as a sweater on a fish. In essence, XML is a written document syntax that just suggests how a system should process it and gain information from it. XML in and of itself is quite simple. However all the "action" really occurs in the next few layers of technology: schema validation, parsing, processing, back-end integration, mapping, messaging, and transformation. In order for the XML to be understandable by a system, a piece of code somewhere on that system has to know what those tags mean and how to do the specific things that that document requires.

XML gives us a standard way to define a document format. That makes writing a parser easier. However, most businesses and industries will create their own versions of the language, which will then require more than just parsing to get an understanding of what that document means. In reality, XML does not provide the capability for seamless data interchange. The truth is that businesses need to agree at some level as to what the information means and how it is to be processed.

It is definitely incorrect to refer to XML as a programming language, since there is no manner in which it instructs a computer how to process information, and can in no way replace languages such as Java or C++. Furthermore, there is no such thing really as "using XML", but rather using some flavor of XML that represents a particular vocabulary of semantic for the exchange at hand.

Some people have tried to replace scripting and programming languages with XML equivalents, but this has proved to be too heavy, verbose, and awkward from a usability perspective. Also, simply put, XML is not a programmatic language. Programs are written in code that's procedural in nature (do this, then this, evaluate this condition, then do that), but XML is a hierarchical, structured format that's better suited towards representing metadata. Another feature of XML is that all content encoded in angle brackets is a string literal. This is good for marking up content, but not appropriate for general-purpose programming, which doesn't really consist of string literals.

The following shows a side-by-side comparison of a standard programming language, C, with a proposed XML equivalent.

```
int main(void) {
    printf("hello");
    return 0;
}
```

*Figure 1: Snippet of Code in "C"*

```
<function name="main" type="int">
  <call function="printf">
    <param type="string">hello</param>
  </call>
  <return value="0"/>
</function>
```

*Figure 2: Proposed XML language equivalent of code snippet*

The benefits in realizing XML as a programming language simply aren't there. XML should be used for what it is: a data format.

**Counterpoint:**

XML was never meant to be a programming language. It was meant to be a better way of representing information in an increasingly linked, online world. XML is a very simple and effective means of data exchange across platforms. The fact that companies are over-hyping XML is a function of market's desire to build unique products and build unique value propositions.

XML is easy to learn, implement, read, and test. It has shortened product development time for most XML-related and data exchange projects that have used available XML standards and technologies.

XML is an effective, portable, easily customized data format that can easily sent over virtually any protocol. While it is just a data format, it can be used for many different purposes, ranging from messaging to RPC. Is XML the end-all, be all for data formats? No, it is simply a data format. The reason XML is so important is that it is an open standard, with an increasingly expanding set of tools that allows plain text to be recognized as data.

The problem is that XML is being applied in every possible scenario even when it is not appropriate. This is primarily a problem in human nature in that people like to use new technologies for all problems, including non-existent problems with existing systems. As they say, "if the only tool you have is a hammer, every problem looks like a nail". The best solution is for appropriate training and experience to show where XML is best applied and leaving programming and scripting languages, configuration files, and relational databases where they are best suited.

## **Con: If XML is so similar to HTML and SGML, why aren't those languages sufficient?**

When people first hear about XML, they often ask why we need another markup language. Everybody's browser supports HTML today, so why create more tags? After all, the basis for XML is the same as that for HTML, namely SGML. Furthermore, why don't we just continue using SGML if it contains all the features that we need?

A lot of people claim that XML echoes many of the earlier claims of HTML itself. They claim that the goals of HTML were to allow for universal processing of information in the context of a browser. After all, HTML uses markup to provide information about embedded content, so why don't we just create new HTML tags that include more data processing directives?

As one person quoted said, "HTML works. It doesn't work like Berners-Lee, et. al. envisaged, but it does work. Scores of applications can read it and make some sense of it. Throw in SQL, which has been around for years, tie them together with your server-scripting engine of choice and, presto, applications anyone can use."

While these arguments are very much misguided in that HTML and XML serve vastly different purposes, a more interesting argument can be made with regards to SGML vs. XML. Since XML is an Internet-enabled, simplified version of SGML, XML doesn't do anything SGML couldn't always do. According to XML detractors, XML is just SGML with a few esoteric, seldom used functions removed that make XML easier to parse. It is true that one can use existing SGML tools to read and write XML, as well as parse and process the documents.

Some of the more sinister arguments have less to do with perceived overlap and redundancy of the XML language than they do with how XML is turning into SGML. The goal of XML was to provide all the benefits of SGML while ridding it of some of the features that detracted from its widespread acceptance and applicability to the Internet, making it more usable for humans and more easily parsed by processors. However, many claim that recent trends by the W3C have complicated the standards in many ways. In particular, there exists some angst in the community towards the XML Schema proposal. It's not that individuals are complaining that XML Schema doesn't meet their needs, but that it is being woven into other proposals in a way that serves to complicate and bloat the core XML standards. They claim that W3C is attempting to weave XML Schema and other technologies such as XPath into other specifications that may or may not be directly relevant to those technologies. This would make adoption of alternate schema or low-level specifications very difficult, if not impossible. These "standards linkages" are causing dependencies that may bring rapid adoption of XML to a grinding halt.

A user who is frustrated by this increasing proliferation and interdependency of standards relates an interesting story: "A vagrant came into a tiny village, begging for food, but nobody would give him anything. So he dropped a stone into a pot of boiling water, declaring that he was making delicious 'stone soup'. People crowded round, anxious to try it, but the vagrant announced that it needed some herbs. Someone brought herbs and threw them in. So it went... after potatoes, carrots, onions, barley, and some meat had been added the stone soup was indeed excellent. Thus with XML. To start with, it was nothing but a subset of SGML - hardly the most exciting standard. Having decided to make it a universal panacea, though, everyone has begun to cooperate in an unprecedented way. It is actually the cooperation that does the work, not XML itself."

There are those that think that XML has become more complex than SGML was, and will necessitate a movement towards a simpler version of XML. In fact, there are already efforts to produce a "Simple Markup Language (SML)" that provides many of the benefits of XML without unnecessary complication. However, it is not clear if this is a symbolic movement or if folks will actually seek to find simplifications of the language. These users seek to create an extensible language that uses optional modules as the manner to expand functionality rather than changing base specifications.

Grumbling about the XML Schema has resulted in competition from other specification organizations for competing schema representations. These include the RELAX specification from ISO and TREX from OASIS. Increasing adoption of these schema may be a watershed occurrence that causes W3C to reevaluate their standards strategy.

These changes are resulting in some problems for XML users and developers. For one, the increasing inter-reliance of specifications is resulting in a rapidly growing learning curve. Even former XML "experts" are having trouble keeping up not only with the new additions to the XML family, but revisions and new interdependencies between them.

XML, who needs it, right?

### Counterpoint:

These claims are ridiculous at first and misguided at best. HTML was created to meet a completely different need than XML, and SGML is far too complicated to be implemented in an easy, credible manner on the Internet.

In the first case, it is clear that XML will not now, nor ever, replace HTML (except of course, the XML based XHTML specification). HTML was designed as a language to present hyperlinked information in a web browser. It has no capability to represent metadata, provide validation, support extensibility by users, or support even the basic needs of e-Business. Fundamentally, the difference is that HTML is meant to be consumed by humans whereas XML is meant for both machine and human consumption. This is made clear by the following comparison of a set of HTML information and its XML counterpart:

```
Contact information:<br>
<b>Ron Schmelzer</b><br>
<b>ZapThink, LLC</b><br>
<b>681 Main Street, Suite 2-11</b><br>
<b>Waltham, MA 02451</b><br>
```

Figure 3: HTML Sample

```
<?xml version="1.0"?>
<contact>
  <firstname>Ron</firstname>
  <lastname>Schmelzer</lastname>
  <company>ZapThink, LLC</company>
  <address>681 Main Street</address>
  <address>Suite 2-11</address>
  <city>Waltham</city>
  <state>MA</state>
  <zip>02451</zip>
</contact>
```

Figure 4: XML sample

So, let's lay the XML vs. HTML issue to rest. As for XML vs. SGML, despite the increasing complexity of the XML specification, there are, and always will be, significant benefits to using XML over SGML.

First, XML permits well-formed documents to be parsed without the need for a DTD. XML is still much simpler and a little more permissive than SGML. The XML specification is very small, includes a bare bones set of features, rather than a bunch of optional features that can make implementation cost difficult to judge, and avoids some of the stigma associated with the SGML name.

XML was created because a direct implementation of SGML on the Internet was difficult. SGML simply did too much. One of SGML's benefits is that it provides significant flexibility for a diverse community of users by providing a wide array of choices, which resulted in a wide range of syntactical variations for documents. This produced a specification that was very difficult for developers to implement. XML 1.0 simplified the specification by eliminating unnecessary flexibility. This resulted in a specification that was

both powerful and easy to implement. The goal was to aim at meeting the majority of users' needs, without aiming to meet all the users' needs.

Those that complain that XML is getting too complicated are forgetting that many of the specifications are not in fact obligatory. XML users can choose to use alternative schema and other technologies as long as they make their choices carefully. The argument that nullifies this concern is that most of this complexity will be hidden behind application interfaces, tools, and middleware. After all, most people will never see, touch, or otherwise interact with schema, namespaces, paths, and linkages. So far, it is mostly developers whining about how the W3C has made their lives more challenging. While they may be correct in their complaints, the fact is that the vast majority of users will never have the opportunity to even see these problems.

XML's advantages, previously outlined in this document, far outweigh the difficulties in developing XML. It still maintains significant differences from SGML, and it is possible that the open processes that helped create XML will likewise limit the effect that underused features will have on the long-term adoption of XML as a technology.

**Con: Every XML format can become a “proprietary” XML format**

The very nature of XML allows one to develop any arbitrary set of tags and data representation. Of course, this is due to spawn hundreds, if not thousands, of data formats that may represent the same piece of information. For example, how many possible ways can there be to represent name and address block information? This "fragmentation" of XML into dozens of similar components can cause any organization that is accepting XML documents as part of their e-Commerce project to be paralyzed by multiple, conflicting formats.

Another, potentially greater problem is the fact that current e-Business XML proposals don't support the breadth and depth of current EDI transactions. EDI supports thousands of specific transactions whereas the most comprehensive e-Business XML specification supports only a few hundred, at most. This inadequacy makes any switch from EDI difficult to swallow, at least for the time being.

Even the presence of XSL doesn't really fully solve the problem, as multiple mappings will need to be maintained for each of the formats. Furthermore, these mappings will need to be maintained in synch with the sending format. If the XML format submitted changes, then the XSL mapping will likewise need to be changed. While external references may make the work of supporting multiple DTDs or Schema a simpler proposition, the sheer number of XML formats, XSL mappings, and externally referenced DTDs make management of XML a nightmare scenario.

Of course, it would all be better if standards were adhered to in a particular industry, but this argument also falls apart. Certain organizations deal with multiple suppliers in multiple vertical industries, each of which may support standards in conflict with other industries. In this case, the management job is a bit simpler, namely supporting a dozen or so formats and XSL stylesheets versus supporting hundreds, but the numbers game is much the same. If it costs time for each XML format to be supported, then the more formats that are accepted as input, the greater the overall cost and complexity will be.

**Counterpoint:**

While the ability to create arbitrarily complex XML documents that conflict with others is a natural outgrowth of XML, the development of increasingly standard horizontal standards "stacks" will simplify matters greatly. An increasing amount of the XML industry is agreeing on a standard way to represent schema, messaging, transport and routing, and service discovery and description. As these layers become increasingly accepted, the job of supporting standards will simplify.

The goal is not to overly simplify the standard, but to provide tools and middleware that eliminates the hassle of supporting dozens of conflicting document definitions. XSL will assist greatly in the task of managing disparate XML document types, and these mappings can be dynamically modified as specifications change. True, it will not be the plug-and-play, "presto-change-o" world promised by the standards bodies, but it will also not be a world of zillions of XML documents floating around with arbitrary exchange types. At some point, it makes sense for both suppliers and vendors to agree on a common way to represent information. The economics work for both sides.

Many also would argue that it is natural, if not beneficial, for so many different document formats to be created. Each XML document is tailored to the specific and best needs for a particular organizations' business logic. It makes sense to even have a completely different internal XML representation than the proposed standard, since no standard will ever meet 100% of a business' needs. As such, a certain amount of transformation between document formats is expected and necessary.

Even if the XML tags are completely different from the representation needed, one of the benefits of XML is that any well-formed document can be parsed – no matter the content contained within. This means that in the worst case, a developer can always parse the "proprietary XML" data using the same technologies as would be applied to "standard XML" data. The only difference is that additional programmatic functionality would be required to make use of the information once it's parsed. The key point here is that XML allows data to be universally navigable, which makes it universally useful.

XML never promised that there would be a universal way to represent data. It promises that there would be a uniform way to parse the information that is delivered. As a commenter in a newsgroup stated, "One should be careful of the fear of new tags. Instead one should embrace the ease of parsing those standards for one's own business needs. Don't be afraid just because there are multiple ways of expressing the same information. Rather, embrace the richness of being able to easily understand them all."

## Con: XML is great for text, but awful for binary data

XML derives its heritage from SGML and shares many of the features of HTML. This lineage has resulted in a format that is excellent for marking up and providing metadata to text documents. This benefit is a double-edged sword. What is good for the text goose is not so great for the binary gander. Simply put, XML is an awful way to represent binary or machine-coded information.

Machine-coded information comes in many forms. Images, video files, sound files, database images, operating system code, and application executables are all represented in binary format since it's not as important for humans to be able to read it as machines. Furthermore, text representation of this information is quite inefficient in size and required processing.

```
<?xml version="1.0"?>
<picture format="base64">DS34JSCSDF02987SDJ3409AS0ASDN3KHSDF.34SD9872...
</picture>
```

*Figure 5: Encoding Binary in an XML document*

So, why would anyone ever put binary data in XML? When they want to include information such as those mentioned above in the context of a document. One way to do so is to encode the information using some ASCII or Unicode-based encoding mechanism. Another method is to actually provide the data values themselves in tags that are representative of their meaning. Each approach is silly, and tries to ram a square binary peg into a round XML hole.

As Sam Blackburn once said on the XML-DEV mailing list: "In short, XML is great for passing around Shakespeare's plays but sucks if you want to send his picture."

### Counterpoint:

Ok, you got me. XML sucks for binary data representation. Of course the question remains: Why would you ever want to represent binary information in an XML document? The answer to this can be quite simple: because humans want to encapsulate all relevant information in a single document.

The use of binary in XML is a perfect example of when XML use is not appropriate. Binary files should stay encoded in binary. If not, the transformation process would require binary documents to be in encoded in ASCII, represented in an XML file, and then unencoded from the XML document to be re-rendered in binary format for processing. These steps are unnecessary. A simple XML link to the binary file would allow transmission of this information by reference and not by value.

```
<?xml version="1.0"?>
<picture uri="http://files.server.net/picture.jpg" />
```

*Figure 6: Binary File Reference in an XML Document*

**Con: XML is a regression from centralized, efficient database systems to inefficient file processing**

A follow-up to the "XML sucks for binary" argument is the notion, expressed by experienced RDBMS gurus, that XML is a major regression from database formats as a data representation format. Their arguments can be boiled down to three major issues: Relational databases have long proved more efficient than hierarchical structures for data storage and representation; XML is a regression from efficient storage mechanisms to file-based processing; and that XML decentralizes data away from powerful central RDBMS data stores.

In the first point, many would argue that hierarchical data formats have been tried before in the 1980's and failed miserably. Their storage requirements and processing inefficiencies precluded wide-spread use. One of the early proponents of relational data models, Codd, pointed out that the main goals of the relational model was to simplify data interchange from the complexities introduced by hierarchical data representations. What is the justification for choosing a more complex, and possibly discredited data model for data exchange, when a majority of commonly used DBMSs employ a simpler, time-tested data model? Most relational database proponents would argue that any true hierarchical tree can be represented and better accessed relationally. The relational data model is actually more general in many respects than the tree structure. In fact, many shoe-horn data to fit the hierarchical requirements of XML.

In the second point, RDBMS systems have long been made efficient in the way that they store, index, and retrieve information. They have evolved into complex, robust, scalable machines that can handle Terabytes of information with ease. Our use of the file-based XML format has in effect, regressed us to the reliance on file processing as a means for processing data. To many in the database community, this is a tremendous step backwards. File processing is inefficient, expensive, and has none of the optimizations that companies have spend decades, billions of dollars, and millions of man hours creating. Are we back to square one with XML storage and retrieval?

To address the final issue, in a database environment, the meaning of data conveyed by any data model is declared by database designers or other designers to the DBMS. The DBMS, then, has the knowledge required to manage data in an efficient manner on behalf of user applications. DBMS systems enforce integrity, perform data manipulation, physically retrieve and update data, and perform a wide range of optimizations. Indeed, the whole point of database management is to centralize these functions in the DBMS, away from file-processing application programs. Why then does XML insist on shipping data AWAY from the primary point of storage so that it can be processed?

**Counterpoint:**

The notion of comparing XML to RDBMS systems in a side-by-side manner is impractical. XML was not created by Bosak et. al. as a means for replacing database functionality. Rather, its goals were to enable a better means for exchange of structured documents that can be used for many purposes, data representation being one.

The needs of users to store and aggregate this data may then necessitate an entirely new class of storage techniques and methodologies. One error commonly made by RDBMS proponents is that they assume that the same sort of information is being represented in XML documents as would be stored in a relational framework. However, this is not always the case.

For data that is hierarchical in nature, many can successfully argue that RDBMS systems are ruefully inadequate. The required disassembly of a structured document into tables and columns, and subsequent reassembly back into an XML format is tremendously inefficiently when compared to the technologies that deal with XML in its native, hierarchical manner. Sure, there are people that are using XML in an inappropriate manner, forcing relational data into a hierarchical format. And in these cases, RDBMS performance will far outweigh XML's supposed efficiencies.

However, if XML is used as intended as a means of storing highly variable, structured data, RDBMS systems simply cannot be competently applied. RDBMS systems require a pre-supposed knowledge of the data structure of information to be inserted. However, XML documents can be extremely variable in their data content and size. Native XML database systems deal with this well, but RDBMS systems would have to jump through hoops to meet these requirements.

So, the basic counter-argument goes: if XML is used to represent hierarchical, structured data, RDBMS systems are not appropriate. If XML is used to represent data that was originally in a relational format, then XML systems become inefficient and inappropriate.

**Con: XML specifications are not yet complete or tested**

Another of the barriers to implementation of XML in a widespread manner is the perception that XML standards are still in development, not yet mature, or not stable enough for widespread deployment.

While this may be an overrated concern, there is some truth to the sentiment. XML has been in existence as a formal recommendation only since 1998, and many of the tools on the market are significantly younger. XML Schema, which adds many necessary features to the language only became a recommendation in May 2001, and the number of tools that support that capability, while growing on a daily basis, are still very young and not quite mature.

Of course, the biggest concern is the stability of the standards. How can users, let alone software tools vendors, support the XML standards if they are constantly shifting? New versions of almost all the standards are in the works. At some point, these standards will need to solidify and give users of all sorts the time they need to properly implement and experiment with the possibilities that these standards offer.

**Counterpoint:**

The simplest counterpoint to the above argument is that the key standards that are important to the development of XML have already been developed. Some would argue that they may even be over-developed. There have been numerous complaints of overly complex, interrelated specifications, and a call to a return to the "simpler days" of XML. As a result, companies should not halt their XML development efforts simply because later specification versions have been halted.

HTML is a good example of a standard that while still very young managed to grow exponentially in its usage and adoption. Sure, browser vendors made changes to their software on a quarterly basis, which resulted in much thrashing in the industry. But such changes have slowed considerably, and the industry is arguably very mature and stable. Won't this be the case with XML as well?

Those companies that are looking to make use of specifications that have yet to be released should operate cautiously in any case, and make sure they their solutions can be easily changed should the specification change suddenly.

## Con: XML DTDs are insufficient for most business and industrial-strength applications, and XML Schema is too complex

Until fairly recently, the main method for verifying validity of documents has been the Document Type Definition (DTD). DTDs have long provided the basic mechanism for guaranteeing that documents contain only a set list of elements and attributes in a specific order and cardinality. It has only been since early 2001 that XML Schema has provided more complex validation mechanisms.

However, DTDs are altogether insufficient to really specify the needs of most business and industry-specific vocabularies. In particular, DTDs aren't capable of specifying data types, bounds for acceptable data ranges, inheritance of schema classes, and support for namespaces. The most egregious of problems with DTDs is that they are not represented in XML syntax. This means that the learning curve for XML includes learning not only the syntax for developing XML documents, but also the separate syntax for creating DTDs.

DTDs are wholly inadequate for businesses since most real-world businesses require that certain values, such as "delivery" be supplied as valid date and "quantities" as positive integers. DTDs only specify that the "price" element should exist, but not what values are considered valid. Also, large DTDs rapidly become unwieldy, necessitating the ability to segment the definition in multiple parts and placing it potentially on multiple systems. This requires the ability to handle inheritance and namespaces, both of which DTDs are wholly incapable of providing.

Despite the inadequacies of DTDs, the elongated development process that resulted in the creation of XML Schema has meant that DTDs have proliferated among early adopters of XML and standards organizations. DTDs frankly are everywhere, and it will take some time before they are replaced with the more robust XML Schema or alternate definition and validation mechanisms.

While we're on the subject of XML Schema, which is supposed to solve many of the problems of DTDs... The long, committee-driven development cycle that resulted in the XML Schema specification followed the "Greatest Common Denominator (GCD) rule". It is an amalgamation of many different ideas and concepts, and has many different features contained within it, solving different needs. Parsers that aim to be fully compliant with XML Schema specification will have to be relatively complex pieces of software to support all the options the specification allows. Of course, it does no good for the parsers to only support a subset of the specification, so the sophistication of those applications will have to rapidly increase in their size, complexity, and power.

However, even XML Schema doesn't contain all the requirements that many XML developers need. XML schemas currently do not support inter-element relationships where the value of one data element requires that another maintain a value. These are the sort of relationships necessitated in e-Business exchanges, and are typically done in a programmatic fashion. XML Schema also does not provide a way to see whether values that look correct actually are correct (as compared against an "authoritative data source").

Another rub on XML Schema is its sheer size and complexity. DTDs are fairly compact formats, while XML Schema, which is based on XML syntax, can easily be three to ten times greater in size.

```
<?xml version="1.0"?>
  <contacts>
    <primary>
      <name>Roland Biggs</cname>
      <email>rbiggs@convict.com</email>
    </primary>
    <person>
      <name>Wye Nott</cname>
      <email>ynot@noname.net</email>
    </person>
    <person>
      <name>Flora Fauna</cname>
```

```

    <email>ff@fn.gov</email>
  </person>
</contacts>

```

Figure 7: XML Example

```

<!ENTITY % record "(name, email)">
<!ELEMENT contacts (primary, person*)>
<!ELEMENT primary %record;>
<!ELEMENT person %record;>

```

Figure 8: Example expressed as a DTD

```

<?xml version="1.0"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="record">
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="email" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="contacts">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="primary" type="record"/>
          <xsd:element name="person" type="record"
            minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>

```

Figure 9: Example expressed in XML Schema

Some also complain that the W3C specifications for XML Schema were developed in a "bubble" without taking into account the rapidly changing environment. In particular, the work of other efforts such as ebXML have modified some of the needs for XML Schema. Some of the newer entrants in the schema space, such as RELAX and TREX have simplified the needs for schema and have met some of these changing landscapes.

The ongoing battle between the necessary features of advanced schema, the entrenchment of DTDs, and the need for a simpler approach to schema development will hopefully result in a delicate balance that pleases the various distinct user communities.

### Counterpoint:

The battle between DTDs, XML Schema, and alternate forms of validation is an unnecessary "holy war". Sure, DTDs are insufficient for many needs, and XML Schema is too complex for other needs, but XML has proved its flexibility in not requiring either format for its ongoing success. Indeed, many standards efforts are supporting multiple schemas in their goal of achieving widespread success. The XML standard will continue to support multiple means for defining schema, as long as the processing tools exist to use those formats.

Furthermore, the complexity of development of Schema is not a general usability issue. It is a development issue. This is a distinct difference that is important to realize. Most people that interact with XML will not be creating Schema, but creating XML documents. Well-defined schema should change at a much slower pace than the XML documents created against it. In addition, the vast majority of these DTD, Schema, or even RELAX-based schema efforts will be created not by humans, but by tools. It makes little sense to continuously hand-code and visually edit a large, unwieldy schema document, whether it is DTD, XML Schema, or RELAX based. The majority of schema will be hidden from both the day-to-day XML users as well as the developers responsible for frequent changes. The only trouble that XML Schema and other

technologies cause is to software developers responsible for coding XML parsers and processors. They don't have the right to complain since complexity actually helps their cause! The more complex the coding requirements, the greater users will depend on tools for their creation and ongoing maintenance. Basically, who cares if an XML Schema is large and unwieldy if it only needs to be edited on an infrequent basis, and even then with help of tools.

Why should necessarily schema be easier to write? Isn't the goal for XML documents to be easy to write, not necessarily schema? Schema will be machine-written, or written once and modified on a less frequent basis than XML documents. The goal isn't necessarily to make schema easier, but to make XML documents more potent and enable real-world business interchange, which the new evolution of XML schema definitely provides.

**Con: XML will never completely replace other technologies, like EDI.**

Even if XML continues to meet and fulfill all its promises as a universal data format, it will not uproot and replace the myriad of technologies currently solving some of the problems that XML aims to solve, albeit in a perhaps better manner.

EDI adoption has been fairly widespread, even though among mainly the larger-sized businesses. The cost of EDI implementation and ongoing maintenance can be measured in the billions in aggregate. Millions of dollars in transactions occur on a daily basis using EDI-mediate messages. It would be very difficult, if not impossible, to uproot all this activity and replace it with exclusively XML-based transactions. These businesses have so much money and time invested in ANSI X12/EDI that they will be fairly slow to adopt a new standard, which would necessitate new processing technology, mapping software, and back-end integration. For them, it would seem that they would need to discard their existing, working technology in favor of an unproven and still immature technology.

For non-EDI systems, XML still presents a "replacement issue". There are plenty of applications where XML is being considered that have existing file formats. These would have to be revised in order to support an XML format. The modification of these applications would require a long, painful process where some developers switch to XML immediately and others wait a while. The replacement would then require users of existing software to upgrade in order to take advantage of the changes. This "propagation of modifications" would result in slow uptake of XML, confusion between XML and non-XML versions of the same application, and possible system conflict.

**Counterpoint:**

First, it should be noted that the primary goal of XML is *not to replace technology, but to augment it*. Those that look at XML and fear it simply because it proposes to solve similar problems as existing technologies, are looking at the gun from the wrong end of the barrel. In the past, people have feared that television would replace radio, and cable would replace television, and the Internet would replace television, radio, and cable. However, in each case the new technology didn't replace the old one, but provided new mechanisms for interaction. In the same way, XML will enable new classes of application and provide new manners to interact with existing systems.

Of course, in the instances where EDI and other technologies provide a weaker business benefit to using XML, there will surely be a replacement occurring. However, these replacements will not happen in a "slash-and-burn" methodology. Rather, XML will be implement when it makes sense, in the best manner possible. Companies will seek to implement XML in the manner that best makes sense to them. Those that insist that XML will replace EDI and other technologies are simply misguided. XML will provide new opportunities to solve existing and emerging problems, and do so in a credible, low-cost, widely supported, and flexible manner.

## **Conclusion**

In the end, we have a collection of arguments in favor and opposed to XML. What is the end result? The end result is that we have a new technology that presents a number of major, significant promises to solve problems that continue to exist. Is XML the answer to these problems? Will it live up to its promises?

We hoped to have provided you a great service in elucidating all the major arguments in favor and against the adoption of XML. We have tried to maintain a balanced approach, favoring neither side, and doing justice to both. If you feel that we have misrepresented a point of view or neglected to mention a major point on either side of the XML "question", please do let us know and we will make our best efforts to represent that point of view.

In the end, XML will face the proverbial Day of Judgment. Which way will the balance tip? As with all technologies, there are many benefits, challenges, trade-offs, and drawbacks. The only answer is the manner in which the technology is used. Adoption and usage of XML is the only thing that will put both the hypesters and nay-sayers at bay.

## **About ZapThink**

Founded in October 2000, ZapThink is an analyst firm focused on the eXtensible Markup Language (XML) and XML Standards, and its adoption by businesses, scientific and academic institutions, and governments. ZapThink provides leading analytical, reporting, and consulting services that help provide complete understanding about a particular technology space by helping clients achieve a complete view of a technology in context with its surroundings.

ZapThink produces and sells the XML.org Standards Report, a quarterly report of over 400 XML standards, as well as a number of other technology and industry-specific reports. Founded in 2000, ZapThink, LLC is headquartered in Waltham, Massachusetts. Its customers include many of the leading Global 1000 firms as well as many emerging businesses. For more information, visit [www.zapthink.com](http://www.zapthink.com)

### **ZAPTHINK CONTACT:**

ZapThink, LLC  
681 Main Street, Suite 2-11  
Waltham, MA 02451  
Phone: +1 (781) 207 8534  
Fax: +1 (786) 524 3186  
[info@zapthink.com](mailto:info@zapthink.com)

## Appendix A: What is XML?

Obviously, since this document is concerned with the various advantages and drawbacks of using XML, we should briefly discuss what exactly is XML. The Extensible Markup Language is a recommendation by the World Wide Web Consortium (W3C) for how to represent structured information in a text-based document. Yes, you heard it correctly; XML is a structured text document, no more no less. Of course, it's not the document itself that matters, but what can be done with it. This latter point will be addressed throughout this document.

XML is a markup language whose roots originate in earlier efforts – a direct descendant of the Standard Generalized Markup Language (SGML) and cousin of the widely accepted and popular HyperText Markup Language (HTML) that currently powers the interface of most web sites. As a markup language, it consists of “elements” encapsulated within angle brackets. These elements supply “metadata” or meaning for the content that is being marked-up. Elements can occur in pairs. When they do, the elements contain the same name reference, but the latter element is prefixed by a slash to indicate that it is the terminal element of the pair. The information between the pair of tags is the “content” that is being marked-up. That is pretty much all that there is to the simple definition of XML. Of course, this means that the elements can be named however you wish, but their meaning is very much specific to the needs of the developer or user.

```
<?xml version="1.0"?>
<MyElement>Hello world</MyElement>
```

*Figure A: XML “Hello World”*

While XML is fairly simple in nature, it can be manipulated in more complex forms. XML elements can be embedded and layered in complicated patterns, and can be modified by “attributes” that attach specific additional meaning and behavior to element tags. In addition, XML documents can take advantage of a number of enhancements such as namespaces.

```
<?xml version="1.0"?>
<!DOCTYPE adXML PUBLIC 'adXML.dtd' 'adXML.dtd'>
<adXML version="1.00" id="234098SDFSF1kj" timestamp="1999-05-02T04:15:47">
  <Header>
    <From>
      <Identity>joesmith@agency1.com</Identity>
    </From>
    <To>
      <Identity>johns@yahoo.com</Identity>
    </To>
  </Header>
  <Request>
    <Agency>
      <Company>
        <Name>Agency1.com</Name>
        <URL>http://www.agency1.com</URL>
      </Company>
      <Contact type="Trafficking">
        <Name>Joe Smith</Name>
        <Title>Campaign Manager</Title>
        <Address>
          <Street>123 N. First</Street>
          <City>San Jose</City>
          <State>CA</State>
          <Country>US</Country>
          <PostalCode>95301</PostalCode>
        </Address>
        <Phone type="mobile">
          <PhoneNumber>
            <CountryCode isoCountryCode="US">011</CountryCode>
            <AreaOrCityCode>415</AreaOrCityCode>
            <Number>808-1832</Number>
          </PhoneNumber>
        </Phone>
        <Email>jsmith@flowerloft.com</Email>
      </Contact>
```

```

    </Agency>
  </Request>
</adxML>

```

Figure B: More Complicated XML Fragment

However, the biggest feature of XML is its ability to provide a means for guaranteeing the validity of a document. This means that one can not only send a document to a receiving party, but also send guidelines with which those documents must comply. For example, a guideline may specify that an XML document should contain only the listed set of elements and attributes in a specific order and given quantities.

```

<!DOCTYPE shirt [
  <!ELEMENT Shirt (Color+,Size+,Price+,Style+,Fabric+,Condition+)>
  <!ELEMENT Color (#PCDATA)>
  <!ELEMENT Size (#PCDATA)>
  <!ELEMENT Price (#PCDATA)>
  <!ELEMENT Style (#PCDATA)>
  <!ELEMENT Fabric (#PCDATA)>
  <!ELEMENT Condition (#PCDATA)>
]>

```

Figure C: Example of a Document Type Definition (DTD)

In the first few years of XML's release, the main means for specifying this validity has been through the Document Type Definition (DTD), a specification based on the format used to specify SGML documents. DTDs use a somewhat arcane and awkward syntax called Extended Backus-Naur Form (EBNF) that allow for a simplistic definition of document content guidelines. However, more recently there have been a number of proposals for improved XML validity specification, culminating in W3C's XML Schema recommendation. These have allowed XML documents to have their syntax and schemata defined using XML format, include simple and complex data typing, support for namespaces, and inheritance capabilities.

```

<?xml version="1.0"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="record">
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="email" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="contacts">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="primary" type="record"/>
          <xsd:element name="person" type="record"
            minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>

```

Figure D: XML Schema sample definition

# XML Expertise Put to Work for You: ZapThink Analysis & Research

Put ZapThink's years of experience and focus in the space of XML to work for you and your organization.

## The ZapThink Mission

The eXtensible Markup Language (XML) will soon be everywhere. It will run your business, your industry, and our world. XML will soon touch every aspect of our lives. Without it, you cannot build a compelling business. With it, you are a lethal competitor.

ZapThink understands XML. Our research is industry and geographically relevant, timely, objective, and respected. ZapThink offers the broadest, most rigorous, and proven research in the industry, delivering the following key services to our clients:

- **Market research** to help you understand the trends in the XML industry and surrounding environment
- **Strategic Advice** to help make XML a success within your organization
- **Product and Vendor analysis** to help you choose the product or vendor that's right for you

ZapThink is a research and analysis firm focused on the eXtensible Markup Language (XML) and XML Standards, and its adoption by businesses, scientific and academic institutions, and governments.

We are more uniquely qualified and better positioned to provide this advice than any other company in the world since we are well-known for our ability to apply detailed and rigorous methodology to research and data sources, while maintaining independence and objectivity.

ZapThink advises hundreds of clients on the usage, adoption, and challenges of applying XML to their business needs. We focus on providing our clients advice on strategic planning and business growth as it applies to the use of XML.

## So What?

***What is XML?***

***What impact will it have on my company?***

***Who else is doing stuff with XML?***

***Why should I care about where XML is heading?***

***What tools and technologies are implementing XML?***

If you find yourself asking these questions, you are not alone. There's a lot of confusion and chaos in the XML industry. Let ZapThink help you sort it out and make sense of the technology that is supposed to be making your life easier, save you money, and help increase your bottom line.

## Our Coverage Areas

ZapThink research and analysis is organized into three major coverage areas, as depicted below:

- **XML Standards** -- what efforts exist to establish a common language, vocabulary, exchange methodology, and protocol for the interaction of multiple business parties?
- **XML Tools and Technology** -- what technologies exist to realize these standards or provide alternate methods for realizing the value of XML?
- **XML Usage and Adoption** -- what is the current state of the XML market and what are the driving trends in its usage and adoption?

## Our Customers

Our customers are those that are looking to use or are already using XML standards, technologies, or tools in their organization. Using a focused and targeted approach, we have served individuals within organizations as diverse as these:

- **Corporate and IT Senior Management**
- **Venture Capital, Private Placement, Public Placement, Executive Committees, and Boards of Directors**
- **Standards Organization management and working groups**
- **Press and Media Relations**
- **Sales and Marketing Management**
- **Procurement, and Finance Executives**
- **IT Project Managers**
- **IT Specialists**
- **Public-Sector Executives**

## XML Standards

In the last few years of the second millenium, a revolution emerged in the way information is represented, transmitted, and used. A veritable explosion of applications have been created that use this data in ways that have never before been seen. Computers that had existed as isolated entities or in closed groups suddenly have the power to interact on a global scale.

And XML is the technology that is making this happen.

In the quest to reign in this explosion, hundreds of XML standards have been created. That's where ZapThink comes in. For the first time in the industry, ZapThink has consolidated and identified every major XML standards effort that is currently shaping the industry. Our research provides companies with the tools they need to evaluate the standards, figure out which direction they are going, identify convergence trends, and help make an evaluation as to whether or not the goals of the standard are in line with the business and product goals of the organization.

## Products and Services

ZapThink has several products and services that help organizations from all industries and of all types evaluate XML and XML standards in the context of their needs.

### **The XML.org XML Standards Report**

The *only* authoritative report covering all 400+ XML standards that have shaped the industry to date, the ZapThink XML Standards Report, licensed by XML.org, is a six-volume, quarterly, comprehensive reference of everything that is going on in the XML Standards landscape. Find every public XML standard in existence with detailed description, analysis, side-by-side comparison, listing of major events, current status, contact information, and XML sample listing. There is no other report on the market like it.

### **Our line-up of ZapThink Research**

ZapThink offers a comprehensive list of research and reports that help support needs around what is happening with XML standards. Our research is up-to-date, rigorous, comprehensive, and objective, and provides critical information on XML standards issues. Please check out our line up of XML industry research for analysis that meets your XML standards evaluation needs.

### **ZapLetter**

The monthly ZapThink newsletter, called ZapLetter, focused on the trends and major events surrounding XML standards development, tools, and adoption. While other sources for XML news exist, no source of information combines the elements together in a manner that keeps you updated with what is actually going on as far as development in all the standards, usage and adoption trends, and tools and technologies in support of XML efforts. Subscribe to the ZapLetter and become even more knowledgeable about the XML industry.

### **ZapThink Analysis and Consulting**

Put ZapThink's expertise in XML and XML standards to work for you! Let us help your organization get a better understanding of what is going on with XML and XML standards. Let us help make XML standards a "best practice" for your organization, whether you are a company looking to make use of XML standards, are already making use of XML technologies, or are a standards organization looking for help in "best practices for Standards Organizations". Let ZapThink's analysts and experts help you make implementing XML standards a success!

## XML Tools & Technology

XML Standards alone are not sufficient. Tools and technology to implement those standards and help make their promises a reality are necessary.

- What tools are available in the space today?
- How can they help me make my business more efficient, cut time, lower costs, increase revenues, or improve collaboration with my partners and associates?
- How do these tools support the XML standards?
- What are other companies in my industry doing with these tools?

Let ZapThink help you answer these questions. Our comprehensive, detailed, and objective analysis of the leading tools on the market will help you make the right decisions.

## **Products and Services**

ZapThink has many products and services available to both organizations seeking tools and technologies as well as tools and technology vendors themselves.

### **ZapAccess**

Are you an XML technology vendor? Are you a business organization that plans to implement XML in a serious manner? If so, take full advantage of ZapThink's leading XML industry analysts. ZapAccess is our comprehensive, yearly subscription service that gives you access to all the research and analysis that ZapThink produces, the ZapLetter, time with ZapThink analysts, feedback into the whitepaper and research development and delivery process, discounts on training, participation in conferences and events, frequent briefing sessions, and the ear of a senior ZapThink analyst. Don't get left behind! Get ZapAccess for your organization today!

### **ZapLetter**

Are you interested in knowing what's going on with XML tools and technologies on a frequent, monthly basis? Would you like to know how the XML technologies landscape is changing on a month-to-month basis? Do you want to keep up with who is adopting which tools and technologies? Then the ZapLetter is for you. Subscribe to the industry's only monthly newsletter focused on where XML is heading from an analysis perspective.

### **ZapThink Tools and Technology Research**

Our timely, up-to-date, comprehensive, and objective research on the various tools and technologies available in the XML space can be found in our comprehensive line-up of ZapThink Research. In our research, you will find focused analysis on the various technologies shaping the industry and the market. Find out which vendors are gaining the most success, and which ones are solving the industry's most critical problems. Look at our current line-up of research and find the report or analysis that best meets your needs!

### **ZapThink Analysis and Consulting: Your Partner**

If XML tools and technologies still have you in a bind, let us do the heavy lifting for you. ZapThink Analysts have the critical experience in order to evaluate which XML technologies and offerings will best meet your needs. Or, if you are an XML Technologies vendor, which industries and markets will be most receptive to your offering. Let us help your company make the best use of XML, in the way that it is both currently offered and the way it is trending in the next few years. ZapThink analysts are available for your needs, on-site, and according to your calendar. Let ZapThink help you make the best XML technologies decisions possible!

## **XML Usage and Adoption**

XML's Achilles' Heel is adoption and usage.

- How is XML being used today?
- How are organizations using or planning to use XML in the next few years?
- What are the global or industrial trends in XML usage and adoption?

- How are tools and technologies impacting the adoption and usage of XML?
- Where is this all heading?

These are the necessary questions for use of XML technology. ZapThink's timely, up-to-date, comprehensive, global, and cross-industry analysis and research digs at the very heart of the matter:

### **WHO IS USING XML AND WHY?**

These are core questions that help to bolster the entire XML industry. Don't enter into XML initiatives with blinders on! Let ZapThink's analysis and research help you make the right decisions.

### **Products and Services**

#### **ZapFlashes**

Interested in getting quick snapshots on what is happening in the XML industry? Make sure you are subscribed to the free email ZapFlashes newsletter that alerts you to the key trends shaping the industry. ZapFlashes supports the efforts of the ZapLetter and ZapThink Research efforts by providing important, time-sensitive, and provocative statistics that help bolster your case for adoption of XML technologies or keep you informed as to what is going on. Keep your finger on the pulse of XML -- subscribe to ZapFlash today!

#### **ZapThink Adoption and Usage Research**

More in-depth analysis and research into XML usage and adoption demands robust, rigorous, comprehensive, timely, and objective research studies. This is where ZapThink's Research comes in to play. ZapThink conducts global, and industry-wide research as to what is going on with XML adoption and usage. We survey the entire market to find out how XML is impacting you today. This information can help make or break your case for use of specific XML technologies. For rigorous XML trends and research, there is only one place to go: ZapThink.

#### **ZapLetter**

Want to know where XML trends in usage and adoption is heading? Subscribe to the ZapThink Newsletter "ZapLetter" today! Featuring key statistics and trends that are not accessible from any other source, the ZapLetter meets your needs in helping you understand where XML is heading. Subscribe today and get your monthly feed on the trends that are shaping XML.

#### **XML Training**

Are you a business-level or strategic-level member of your organization, and need a better understanding of how XML will impact your business? Are you interested in having your management or development teams become more knowledgeable in the areas where XML will impact development and business the most? Then ZapThink's Management-level XML Training fits the bill. ZapThink will deliver a focused, customized training curriculum that will inject the best practices and knowledge of XML into your organization. Let ZapThink's experience help you

## Putting it all together: What does it mean for you?

ZapThink is all about supporting you and your organization, and is committed to helping you successfully meet and solve the challenges that face your organization when it comes to implementing and using XML, XML Standards, and XML technologies. We are successful when we understand your problems and bring to bear our high level of quality, professional, relevant, and timely analysis and research skills for the benefit of your organization.

Our expertise is in helping clients navigate through the ever-shifting waters of XML, solve their business needs using the technology and standards, and reap real business rewards from the understanding of how XML impacts the bottom line. Our proven ability to understand the nuances and realities of XML, combined with our focused, detailed, timely, and comprehensive research and analysis, makes ZapThink the right place to go for all your XML analysis needs.

## About ZapThink

Founded in October, 2000, ZapThink is an analyst firm focused on the eXtensible Markup Language (XML) and XML Standards, and its adoption by businesses, scientific and academic institutions, and governments. ZapThink provides leading analytical, reporting, and consulting services that help provide complete understanding about a particular technology space by helping clients achieve a complete view of a technology in context with its surroundings.

The ZapThink staff has many years of experience in the field of XML Standards. Its analysts have sat on the working group committees for standards bodies such as RosettaNet, UDDI, CPExchange, ebXML, EIDX, and CompTIA.

ZapThink is located in Waltham, Massachusetts. The mailing address is listed below:

ZapThink LLC  
681 Main Street  
Suite 2-11  
Waltham, MA 02451  
Phone: 781-207-8534  
Fax: 786-524-3186

Our United Kingdom fax number is: +44-870-135-5156

[www.zapthink.com](http://www.zapthink.com)  
[info@zapthink.com](mailto:info@zapthink.com)

