Structured authoring and XML

Sarah O'Keefe
Chief Executive Officer

Alan Pringle
Chief Operating Officer

info@scriptorium.com

SCRIPTORIUM

Introduction

Structured authoring and XML represent a significant paradigm shift in content creation. Implementing structured authoring with XML allows organizations to enforce content organization requirements. The addition of hierarchy and metadata to content improves reuse and content management. These benefits, however, must be weighed against the effort required to implement a structured authoring approach. The business case is compelling for larger writing organizations; they will be the first to adopt structured authoring. Over time, improvements in available tools will reduce the cost of implementing structured authoring and make it affordable for smaller organizations.

What is structured authoring?

Structured authoring is a publishing workflow that lets you define and enforce consistent organization of information in documents, whether printed or online. In traditional publishing, content rules are captured in a style guide and enforced by (human) editors, who read the information and verify that it conforms to the approved style. A few simple examples of content rules are as follows:

- A heading must be followed by an introductory paragraph.

- A bulleted list must contain at least two items.

- A graphic must have a caption.

In structured authoring, a file—either a document type definition (DTD) or a schema—captures these content rules. Authors work in software that validates their documents; the software verifies that the documents they create conform to the rules in the definition file.
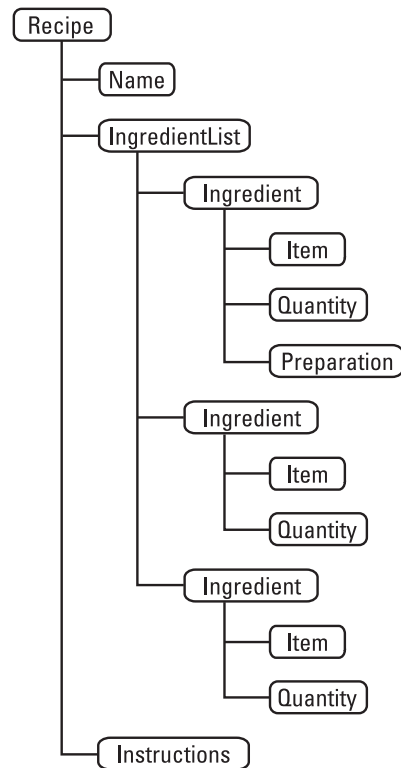
Consider, for example, a simple structured document—a recipe. A typical recipe requires several components: a name, a list of ingredients, and instructions. The style guide for a particular cookbook states that the list of ingredients should always precede the instructions. In an unstructured authoring environment, the cookbook editor must review the recipes to ensure that the author has complied with the style guideline. In a structured environment, the recipe structure requires and enforces the specified organization.

Elements and hierarchy

Structured authoring is based on elements. An *element* is a unit of content; it can contain text or other elements. You can view the hierarchy of elements inside other elements as a set of nodes and branches.

Elements can be organized in hierarchical trees. In a recipe, the ingredient list can be broken down into ingredients, which in turn contain items, quantities, and preparation methods.

Figure 1: Recipe hierarchy

```
Recipe
    Name
    IngredientList
            Ingredient
                    Item
                    Quantity
                    Preparation
            Ingredient
                    Item
                    Quantity
            Ingredient
                    Item
                    Quantity
    Instructions
```

The element hierarchy allows you to associate related information explicitly. The structure specifies that the IngredientList element is a child of the Recipe element. The IngredientList element contains Ingredient elements, and each Ingredient element contains two or three child elements (Item, Quantity, and optionally Preparation). In an unstructured, formatted document, these relationships are implied by the typography, but unstructured publishing software (a word processor or desktop publishing application) does not capture the actual relationship.

- In structured documents, the following terms denote hierarchical relationships:

- Tree—The hierarchical order of elements.

- Branch—A section of the hierarchical tree.

- Leaf—An element with no descendant elements. Name, for example, is a leaf element in Figure 1.

- Parent/child—A child element is one level lower in the hierarchy than its parent. In Figure 1, Name, IngredientList, and Instructions are all children of Recipe. Conversely, Recipe is the parent of Name, IngredientList, and Instructions.

- Sibling—Elements are siblings when they are at the same level in the hierarchy and have the same parent element. Item, Quantity, and Preparation are siblings.

Element attributes

You can store additional information about the elements in attributes. An attribute is a name-value pair that is associated with a particular element. In the recipe example, attributes might be used in the top-level Recipe element to provide additional information about the recipe, such as the author and cuisine type.

Attributes provide a way of further classifying information. If each recipe has a cuisine assigned, you could easily locate all Greek recipes by searching for the attribute. Without attributes, this information would not be available in the document. To sort recipes by cuisine in an unstructured document, a cook would need to read each recipe.

Recipe
Author = "John Doe"
Cuisine = "American"

Formatting structured documents

To format structured documents, you associate formatting with particular elements or element sequences. Such formatting is usually highly automated; once an author assigns elements to content, the formatting is implemented automatically to create the final output files.

What is XML?

Extensible Markup Language (XML) defines a standard for storing structured content in text files. The standard is maintained by the World Wide Web Consortium (W3C).[1]

XML is closely related to other markup languages, such as Standard Generalized Markup Language (SGML). Implementing SGML is an enormous undertaking. Because of this complexity, SGML's acceptance has been limited to industries producing large volumes of highly structured information (for example, aerospace, telecommunications, and government).

XML is a simplified form of SGML that's designed to be easier to implement.[2] As a result, XML is attractive to many industries that create technical documents (including parts catalogs, training manuals, reports, and user guides).

Note: As of 2017, XML has almost completely displaced SGML.

XML syntax

XML is a markup language, which means that content is enclosed by tags. In XML, element tags are enclosed in angle brackets:

```
<element>This is element text.</element>
```

A closing tag is indicated by a forward slash in front of the element name.

---

[1] Detailed information: http://www.w3.org/XML/

[2] SGML vs. XML details: http://www.w3.org/TR/NOTE-sgml-xml-971215

Attributes are stored inside the element tags:

```
<element my_attribute="my_value">This is element text.</element>
```

XML does not provide a set of predefined tags. Instead, you define your own tags and the relationships among the tags. This makes it possible to define and implement a content structure that matches the requirements of your information. Figure 2 shows an XML file that contains a recipe.

Figure 2: Recipe XML

```
<Recipe Cuisine = "Italian" Author = "Unknown">
  <Name>Marinara Sauce</Name>
  <IngredientList>
    <Ingredient>
      <Quantity>2 tbsp.</Quantity>
      <Item>olive oil</Item>
    </Ingredient>
    <Ingredient>
      <Quantity>2 cloves</Quantity>
      <Item>garlic</Item>
      <Preparation>minced</Preparation>
    </Ingredient>
    <Ingredient>
      <Quantity>1/2 tsp.</Quantity>
      <Item>hot red pepper</Item>
    </Ingredient>
    <Ingredient>
      <Quantity>28 oz.</Quantity>
      <Item>canned tomatoes, preferably San Marzano</Item>
    </Ingredient>
    <Ingredient>
      <Quantity>2 tbsp.</Quantity>
      <Item>parsley</Item>
      <Preparation>chopped</Preparation>
    </Ingredient>
  </IngredientList>
  <Instructions>
    <Para>Heat olive oil in a large saucepan on medium. Add garlic and hot
red pepper and sweat until fragrant. Add tomatoes, breaking up into smaller
pieces. Simmer on medium-low heat for at least 20 minutes. Add parsley,
simmer for another five minutes. Serve over long pasta.
    </Para>
  </Instructions>
</Recipe>
```

XML is said to be well-formed when basic tagging rules are followed. For example:

• All opening elements have a corresponding closing element, and empty elements use a terminating slash:

```
<element>This element has content</element>
<empty_element />
```

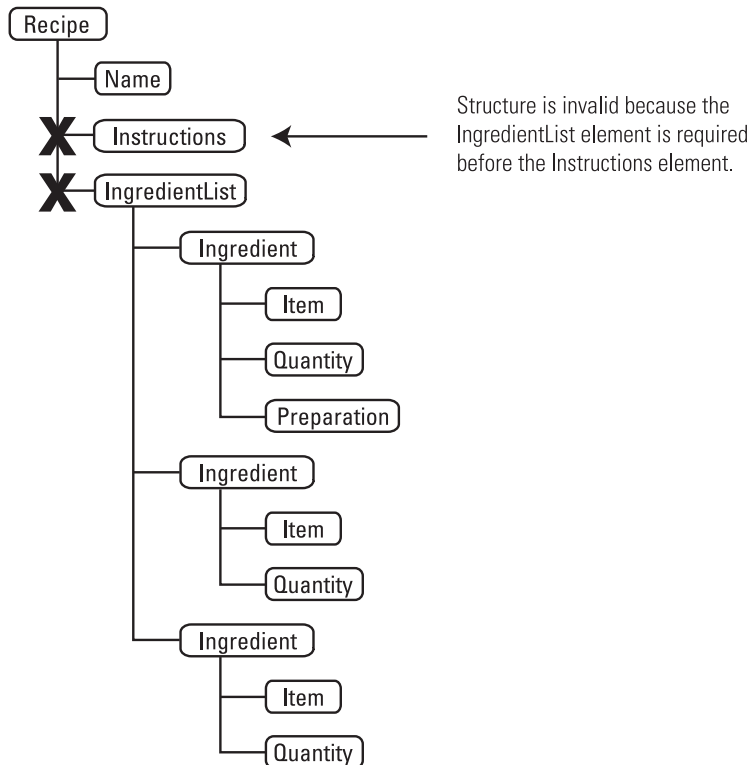• Attribute information is enclosed in double quotes:

```
<element attribute="name">This is a legal attribute.</element>
<element attribute=name>This is not well-formed.</element>
```

- Tags are nested and do not "cross over" each other

```
<element>This is <strong>correct.</strong></element>
<element>This is<strong>not correct.</element></strong>
```

XML is said to be *valid* when the structure of the XML matches the structure specified in the structure definition. When the structure does not match, the XML file is *invalid.* In the following example, the structure is invalid because the IngredientList element is in the wrong location.

```
Recipe
    Name
X   Instructions          ← Structure is invalid because the
X   IngredientList            IngredientList element is required
        Ingredient            before the Instructions element.
            Item
            Quantity
            Preparation
        Ingredient
            Item
            Quantity
        Ingredient
            Item
            Quantity
```

Entities

> **Note:** As of 2017, entities are rarely used in XML content.

An XML entity is a placeholder. Entities allow you to reuse information; for example, you could define an entity for a copyright statement:

```
<!ENTITY copyright "Copyright 2017 Scriptorium Publishing Services, Inc. All
  rights reserved.">
```

To reference the entity, you refer to the entity name:

```
&copyright;
```

The entity text is displayed instead of the entity name:

```
Copyright 2017 Scriptorium Publishing Services, Inc. All rights reserved.
```

Storing common information in entities lets you make a change in one location (the entity definition) and have the change show up everywhere that references the entity.

Entities are also used to include information that can't be easily rendered as text. Graphics, for example, can be referenced as entities. In the following example, the entity definition contains the entity name, graphic file name, and file type:

```
<!ENTITY my_image SYSTEM "image.gif" NDATA gif>
```

In the XML file, a Graphic element references this entity:

```
<Graphic entity = "my_image" />
```

How are XML and structured authoring related?

Structured authoring is a concept. XML is a specification that lets you implement structured authoring using plain text files. In the past, most structured authoring implementations were based on SGML; today, XML is the standard. The terms XML and structured authoring are often used almost interchangeably.

Unlike SGML, XML is widely used outside the technical publishing world, especially for data interchange and web services applications.

Defining structure in XML

In XML, you define your structure using either a DTD or schema. In either case, you specify elements and how they are related to each other. For example, a Recipe element definition might read as follows in a DTD:

```
<!ELEMENT Recipe (Name, History?, IngredientList, Instructions)>
```

In an XML schema, the definition is itself an XML document. For the Recipe element, a simplified Recipe definition would read as follows:
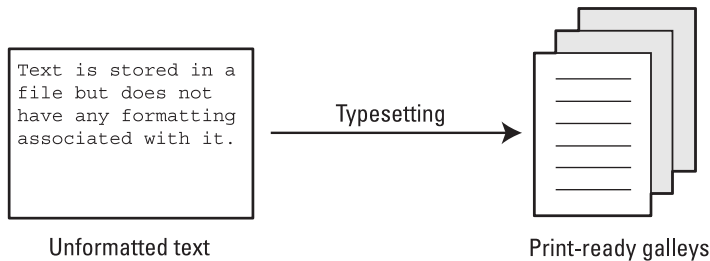
```
<xsd:complexType name="Recipe">
     <xsd:sequence>
       <xsd:element name="Name" type="xsd:string"/>
       <xsd:element name="History" type="xsd:string" minOccurs="0"
 maxOccurs="1">/>
       <xsd:element name="IngredientList" type="xsd:string"/>
       <xsd:element name="Instructions" type="xsd:string"/>
     </xsd:sequence>
    <xsd:complexType>
```

Once you define the structure, authors create documents that comply with the structure. At a bare minimum, this allows you to specify, for instance, that the list of ingredients in a recipe must occur before the instructions.

Schema are especially useful in XML-based programming applications, where they allow you to validate and restrict data inside the structure. DTDs are more common in publishing applications, partly because of the legacy with SGML. In long, technical documents that consist mostly of paragraphs, the validation provided by schema does not add a significant amount of value.

The impact of structured authoring on a publishing workflow

A generation ago, technical writers began to make the transition from typewriters to computer-based writing. Initially, authors stored text in word-processing files, but formatting was done in a separate typesetting operation.

```
Text is stored in a
file but does not
have any formatting
associated with it.
```
Typesetting →

Unformatted text   Print-ready galleys

Next, the transition from dedicated word processing equipment to personal computers led to word processing software with the added ability to control formatting with embedded formatting codes. Authors learned how to write and format their documents.

```
[font Arial]
[size 12 points]
Formatting codes let
you embed formatting
instructions [bold]
directly [bold] in the
document.[HRT]
```

Text with formatting codes

Formatting codes were soon grouped into paragraph styles or tags. Instead of specifying font, font size, alignment, and the like, the author specified a style code, which contained a group of formatting settings.

```
[style = Heading1]
Understanding styles
[style = Normal]
Paragraph styles let you
store formatting settings
as a group.
```
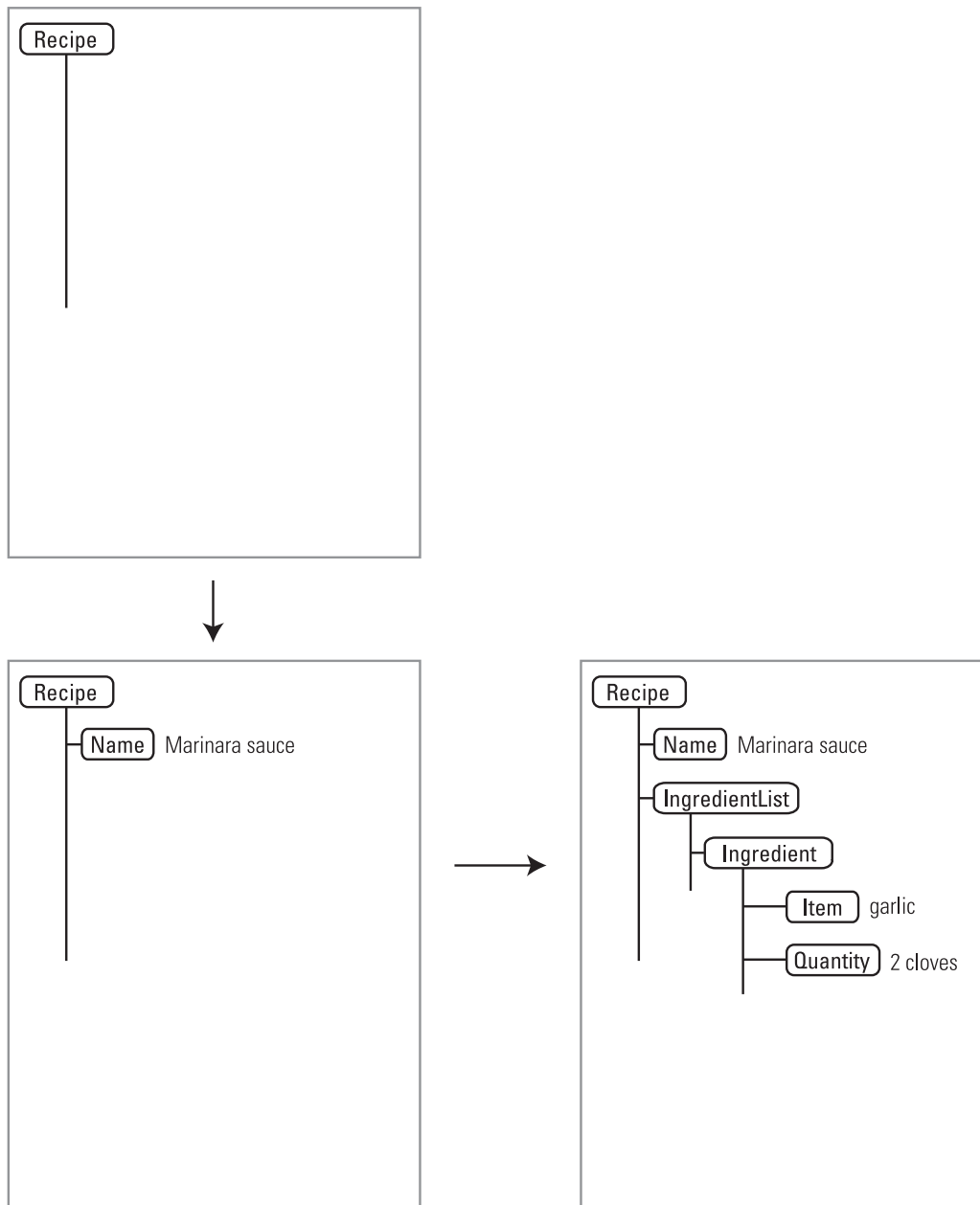
Text with styles

With paragraph style sheets, a template designer could define the look and feel of documents for an entire workgroup by setting up a formatting template.

In some environments, templates are enforced strictly; in others, individual authors are allowed to customize formatting to suit their document and their personal preferences. When formatting and content development are separated, this special formatting becomes impossible.

In a structured authoring environment, authors create documents by assembling elements and text in an order permitted by the structure definition document (Figure 3 on page 9).
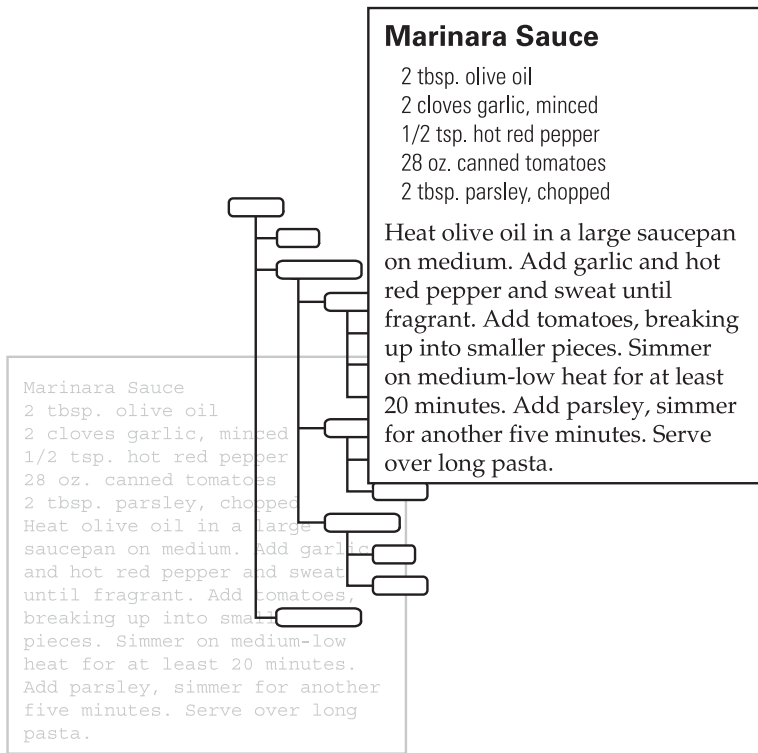
You might think of structured authoring as being similar to template-based authoring with a strict template. Authors do not assign formatting; formatting is automatically assigned based on the structure of the document. Each output format has its own formatting specification.

Figure 3: Structured authoring from the author's point of view

Changing perceptions

XML and structured authoring result in a completely different way of looking at information. Instead of the familiar page- and paragraph-based metaphor, structured authoring requires that authors consider information as a hierarchy with a separate formatting layer.

Marinara Sauce

2 tbsp. olive oil
2 cloves garlic, minced
1/2 tsp. hot red pepper
28 oz. canned tomatoes
2 tbsp. parsley, chopped

Heat olive oil in a large saucepan on medium. Add garlic and hot red pepper and sweat until fragrant. Add tomatoes, breaking up into smaller pieces. Simmer on medium-low heat for at least 20 minutes. Add parsley, simmer for another five minutes. Serve over long pasta.

```
Marinara Sauce
2 tbsp. olive oil
2 cloves garlic, minced
1/2 tsp. hot red pepper
28 oz. canned tomatoes
2 tbsp. parsley, chopped
Heat olive oil in a large
saucepan on medium. Add garlic
and hot red pepper and sweat
until fragrant. Add tomatoes,
breaking up into smaller
pieces. Simmer on medium-low
heat for at least 20 minutes.
Add parsley, simmer for another
five minutes. Serve over long
pasta.
```

A document's formatting can imply a certain structure—for example, a large, sans-serif font often indicates an important heading—but unstructured files do not describe how paragraphs are related to each other (Figure 4). XML makes it possible to encode structure into a document explicitly (Figure 5 on page 11).

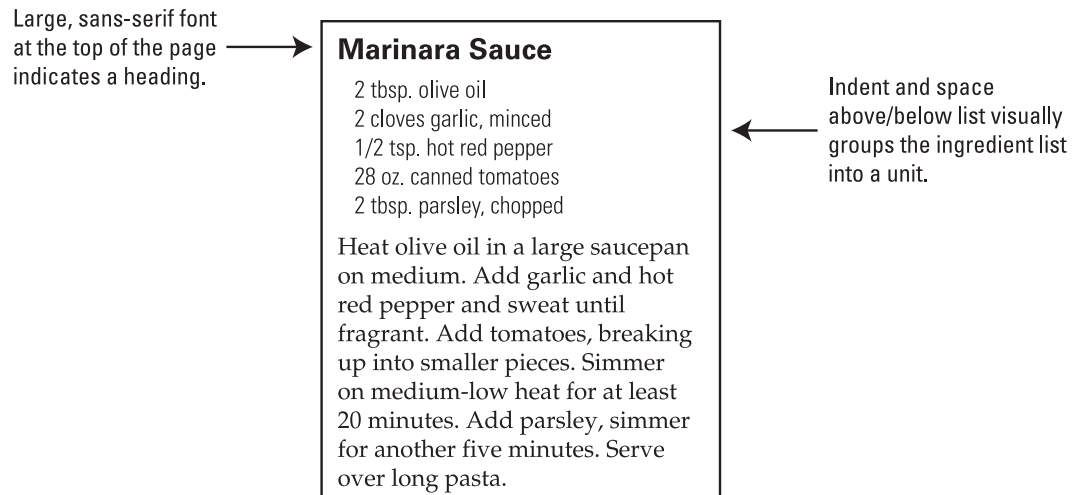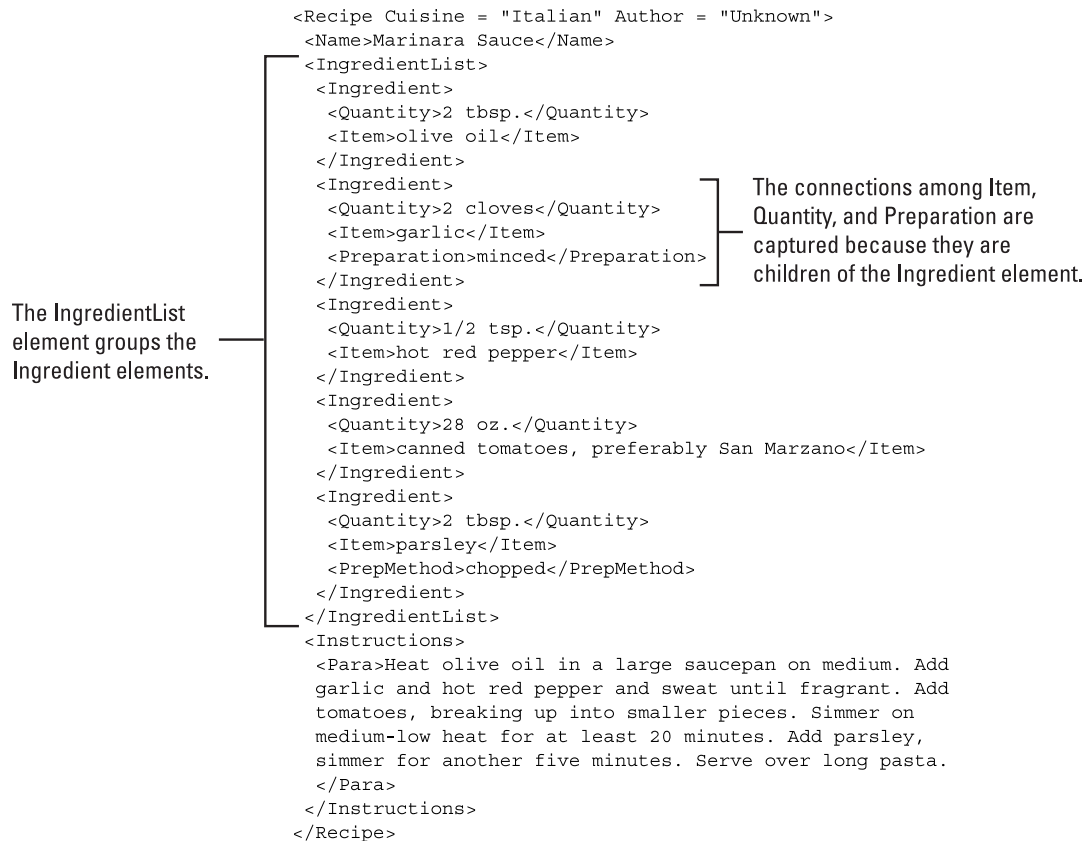*Figure 4: Formatting can imply structure*

Large, sans-serif font at the top of the page indicates a heading.

Marinara Sauce

2 tbsp. olive oil
2 cloves garlic, minced
1/2 tsp. hot red pepper
28 oz. canned tomatoes
2 tbsp. parsley, chopped

Heat olive oil in a large saucepan on medium. Add garlic and hot red pepper and sweat until fragrant. Add tomatoes, breaking up into smaller pieces. Simmer on medium-low heat for at least 20 minutes. Add parsley, simmer for another five minutes. Serve over long pasta.

Indent and space above/below list visually groups the ingredient list into a unit.

Figure 5: XML captures structure explicitly

```
<Recipe Cuisine = "Italian" Author = "Unknown">
 <Name>Marinara Sauce</Name>
 <IngredientList>
  <Ingredient>
   <Quantity>2 tbsp.</Quantity>
   <Item>olive oil</Item>
  </Ingredient>
  <Ingredient>
   <Quantity>2 cloves</Quantity>
   <Item>garlic</Item>
   <Preparation>minced</Preparation>
  </Ingredient>
  <Ingredient>
   <Quantity>1/2 tsp.</Quantity>
   <Item>hot red pepper</Item>
  </Ingredient>
  <Ingredient>
   <Quantity>28 oz.</Quantity>
   <Item>canned tomatoes, preferably San Marzano</Item>
  </Ingredient>
  <Ingredient>
   <Quantity>2 tbsp.</Quantity>
   <Item>parsley</Item>
   <PrepMethod>chopped</PrepMethod>
  </Ingredient>
 </IngredientList>
 <Instructions>
  <Para>Heat olive oil in a large saucepan on medium. Add
  garlic and hot red pepper and sweat until fragrant. Add
  tomatoes, breaking up into smaller pieces. Simmer on
  medium-low heat for at least 20 minutes. Add parsley,
  simmer for another five minutes. Serve over long pasta.
  </Para>
 </Instructions>
</Recipe>
```

The connections among Item, Quantity, and Preparation are captured because they are children of the Ingredient element.

The IngredientList element groups the Ingredient elements.

Adding metadata to documents

Metadata is information that describes or classifies other information. A word-processing document usually contains basic metadata, such as the document's title, author, and keywords.

Structured authoring supports metadata with elements and attributes. Element names themselves can provide metadata; for example, naming elements GlossaryTerm and GlossaryDefinition encapsulates a lot of information about the elements' content. Attributes provide a way to label elements with additional information. Once the attributes are set up, you can then include, exclude, or process information based on the value of the attributes.

In structured authoring, you can assign metadata to elements in a document. With metadata, you label information with identifiers, such as:

- Version

- User level

- Revision date

- Author

Element attributes give you much finer control over metadata than the basic file-level information you can store in word-processing documents.

Workflow options

XML and structured authoring do not provide an actual workflow; they must be incorporated into a complete workflow. Before establishing a structured publishing workflow, you should consider the following tasks:

- Defining content sources

- Establishing content repositories

- Implementing content reuse

- Delivering formatted output

Defining content sources

It's important to examine existing content to establish how it is currently developed and how it will benefit from structure. Other questions include the following:

- Can all of the content be stored in a single location?

- Is it necessary to keep different versions of the same content, or can information come from a single source?

- Who develops the content? How often is it updated?

- Are there dependencies between different sets of content?

The following table shows a simplified audit of content:

Information product	Current tool	Benefit from structure?	Dependencies	Updated?
User guide	FrameMaker	Yes		Twice a year
Training manuals	PowerPoint	Yes	Uses info from user guides	Quarterly
Online help	Flare	Yes	Uses info from user guides	Monthly
Release notes	Word	No		No
Marketing white papers	Word	No		No

Establishing content repositories

A content repository or database is *not* required to work in XML. However, a content repository makes it possible to manage content modules, which allows you to do the following:

- Search content by elements and attribute

- Locate content created by a specific author

- Locate content by topic

- Identify content chunks that are being used in multiple locations

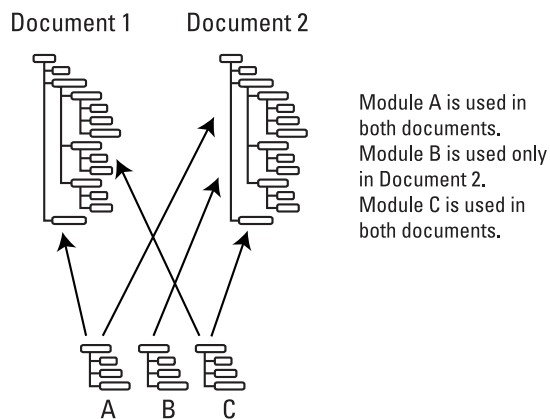- Extract chunks that match certain criteria

Because XML is a text format, it works very well with content repositories.

Structured authoring improves consistency across documents. This makes it easier to manage them in a content repository. Content can be automatically chunked at specified element levels, which makes content reuse easier.

Implementing content reuse

Content reuse, or single sourcing, doesn't require an XML-based workflow. In XML, though, it's easier to enforce the consistency that's required to make content reuse work. Content reuse means that you develop a particular chunk of information once, and then use it wherever it's needed. Reuse can occur across media—for example, a chunk of content is used in both the printed manual and in the online help for a product. In other cases, you might write a chunk of information that's needed in several different printed books. Reusing that chunk minimizes maintenance and ensures consistency across all of the information products.

Delivering formatted output

Structured authoring separates structure and formatting; this provides both the greatest advantage and the greatest challenge in the structured environment.

Authors are accustomed to working in a visual environment. Requiring them to work solely with tags is impractical, yet providing an approximation of the final output's appearance will bias them toward a particular medium. For example, an authoring environment that looks something like a printed page makes it more difficult to consider how content will function in

an online help format. Figure 6 outlines how structured information is processed to produce the final deliverable documents in specific formats.

As of 2017, there are numerous authoring and publishing tools available for XML, both commercial and open-source.

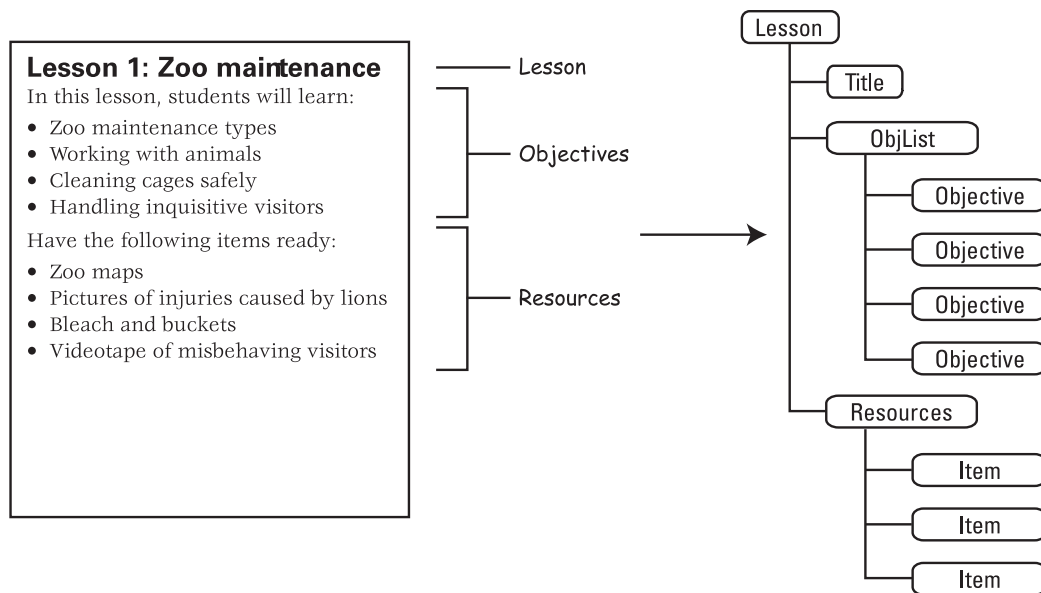*Figure 6: Processing structured documents to produce final output*

Roles and responsibilities

The roles and responsibilities in a typical publishing group change when structured authoring is implemented. This section explains how traditional roles change and describes the new role of the information architect.

Note that in a small group, one person may hold any or all of these roles.

Information architect

The information architect defines and implements document structure. The information architect must identify information types and establish their required structure. For example, an information architect would build a structure for a company's training manuals.

Template designer

The template designer is responsible for establishing the look and feel of content deliverables, such as books, online help, e-learning, and so on. In traditional desktop publishing, the template designer is usually a tools expert who can create templates in the appropriate publishing tools. In a structured authoring environment, the designer might also be asked to learn Extensible Stylesheet Language (XSL) and XSL Formatting Objects (XSL-FO) to create HTML, PDF, and other output from the XML files.[3]

Writer

In a structured workflow, writers, as always, create content. In the 1990s, writers often were asked to take on additional formatting and publishing responsibilities; in a structured workflow, these tasks are generally automated. The information architect establishes the

---

[3] More information about XSL and XSL-FO: http://www.w3.org/Style/XSL/

overall structure of the documents; the template designer implements a look and feel that is automatically assigned based on the structure of the document.

Many writers who are new to structure are uncomfortable with the perceived lack of control over the final document. They have become accustomed to "tweaking" the final output to make it look right. Any implementation of a structured workflow must anticipate some resistance and perhaps even outright hostility from some writers.[4]

This resistance seems misplaced, though. Instead of wrestling with formatting problems, writers can focus on content and organization—typically a better fit for writers' skills and interests than desktop publishing. After the initial transition and learning curve, working within a structure increases writer productivity and improves the quality and consistency of the final output.

Technical editors

By enforcing correct structure during content development, a structured workflow eliminates the need for editors to check a document for structure. Instead, editors can focus on word choice, grammar, and overall organization. By automating some of the most tedious parts of the editing job, a structured authoring environment makes it possible for editors to do a more thorough edit in the same amount of time.

Editors are also uniquely positioned to assist with structure implementation. Technical editors see more of a total document library than any other member of a publishing team (with the possible exception of production editors). Because of this familiarity with the overall documentation set, editors are excellent resources to assist in establishing an information architecture.

Editors may also have the skills to establish the needed *taxonomy* for metadata. Taxonomy is a classification system; in a structured workflow, classifying includes defining element names and hierarchy, which elements need attributes, and what values those attributes could have.

> **Note:** (2017) Technical editing jobs are few and far between. Some organizations have resorted to "peer editing," and others have eliminated the technical editing role.

Production editors

In many companies today, writers and editors handle production tasks, but there are a few documentation teams that still have production editors. With formatting generated automatically by the structure of a document, the workload for these production editors should decrease. Many production editors will refocus their efforts on the transformation part of the workflow. Instead of correcting formatting errors after the fact, they will get involved in defining the transformation files that assign formatting based on structure. Production editors will verify that output formatting is working correctly.

> **Note:** (2017) Production editing jobs are almost extinct.

---

[4] Wider resistance may indicate the new structure does not accommodate all types of content. A thorough analysis of multiple document types before implementation will minimize this problem. However, it is important to have a change process in place to handle revisions to the structure.

A difficult transition?

The transition from "free-form" writing to structure can be difficult. Just as some writers dislike working in a template-driven environment where formatting is constrained, some dislike the regimentation of structured authoring.

Structured authoring offers the business organization compelling advantages, including improved consistency and increased productivity because manual editing and formatting time are decreased. The widespread implementation of structured workflows will likely result in structure being used to deliver information in ways we have not yet even anticipated. It is indisputable, though, that structured information is more valuable than unstructured information. These advantages must be weighed against the arguments from the writers that writing in a structured environment is "less interesting."

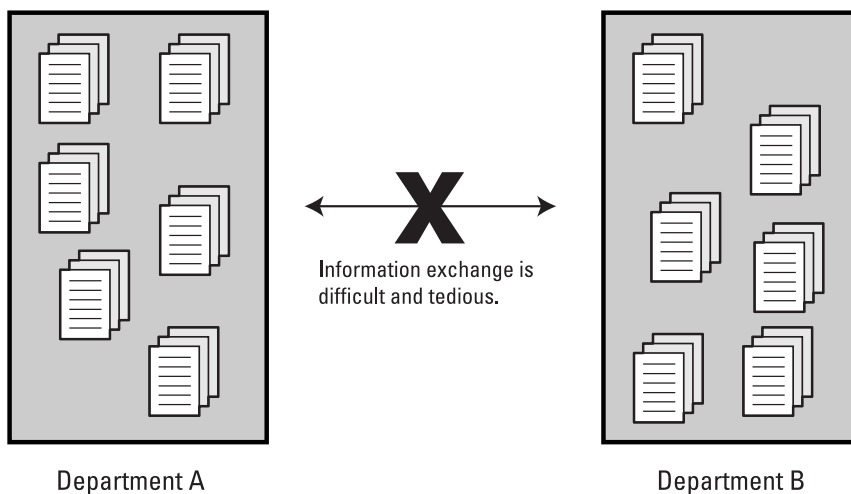# Developing a business case for structured authoring and XML

Not every content-creation group will benefit from structured authoring and XML. Sometimes, the expense of implementation outweighs the benefits realized, especially in smaller groups with a smaller amount of content.

There are a number of imperatives that lead to implementation of structured authoring and XML. The following are some of the most common scenarios:

- Enabling content exchange between incompatible applications

- Extracting information from databases for publication

- Reducing content duplication and reusing information

- Extracting information based on structure and metadata

- Automated formatting in all languages

- Reducing author learning curve

- Improving compliance with required document structure, especially in regulated industries
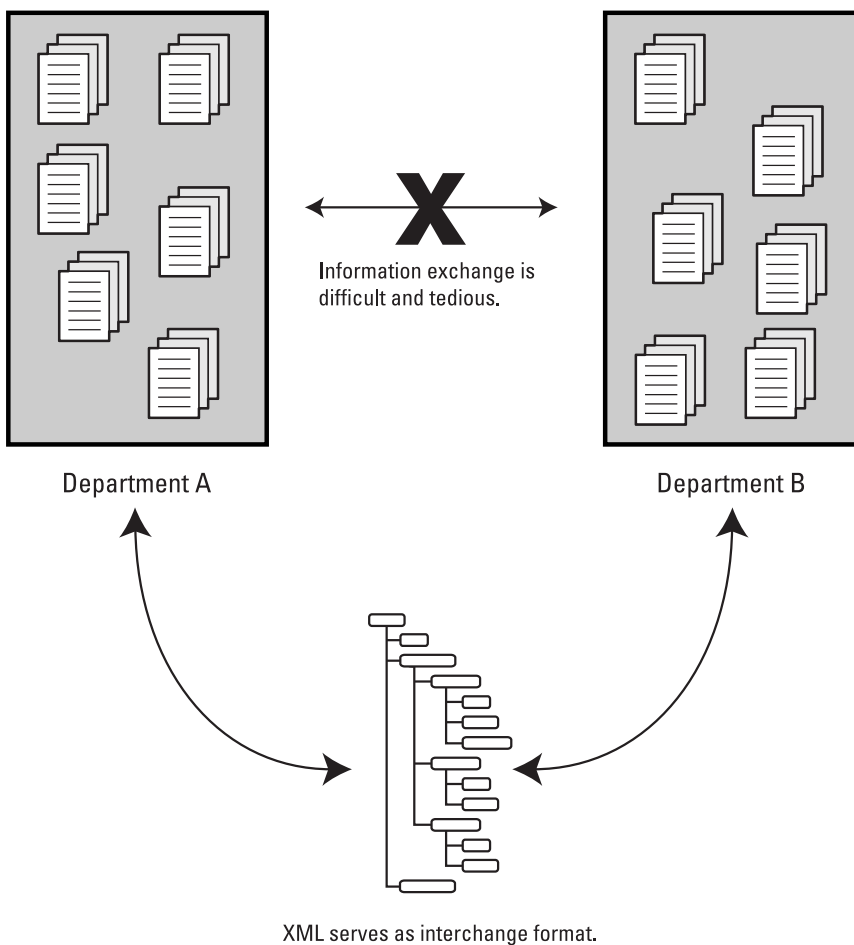
Enabling content exchange

XML is platform- and vendor-neutral, which makes it an excellent choice as an intermediate format. It is quite common in a single company for two departments to standardize on different, incompatible publishing tools. As a result, the information developed in one department cannot be reused in another department without extensive manual conversion and reformatting work. This leads to "content silos," where each department owns a separate, private set of information, often with significant amounts of content duplication.

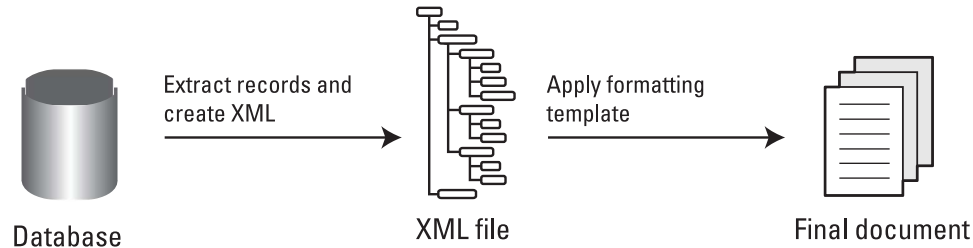Department A   Department B

Structured authoring and XML can eliminate this silo mentality without necessarily forcing either group to implement the preferred software tools of the other group. Each group authors in its preferred application, and then exports to XML for interchange. Intensive coordination is required to ensure that the structures used by each group are compatible.

To make this system work, each group must use a publishing tool that supports XML import/export.

Department A   Department B

XML serves as interchange format.

Extracting information from databases for publication

XML provides a useful intermediate format for content that's exported from a database. Most commonly, database publishing is used for parts catalogs, directories, and similar large data sets. The records are extracted from the database and marked up as XML; the XML is then processed to produce the final output.

Traditionally, database publishing has required customized, application-specific solutions. XML offers a generalized and significantly less expensive approach, which better separates the data generation task from the output formatting task.

Extracting information based on structure and metadata

Once information is structured and stored with metadata attached to it, it becomes much easier to search for specific information. Consider a structured environment in which each major topic has the following attributes:

- Author name
- Revision date
- Product/topic
- User level
- Platform

Based on these attributes, you could perform a search that extracts all of the topics written in the past year that are Windows specific and for administrators.

Automated formatting in all languages

In a structured environment, formatting is handled automatically based on the structure. "Formatting by rule" greatly improves consistency across a document set—authors or production editors are not required to remember, for example, that in a list of bullets, the first bullet gets a special paragraph tag. Instead, the software applies these types of formatting rules automatically.

Automated formatting reduces the overall effort of creating a document. For organizations that produce content in multiple languages, the cost savings are multiplied. As of 2017, the need for efficient localization is one of the most common business justifications for moving into XML.

Reducing author learning curve

Instead of learning to format documents using a specific publishing tool, writers focus on creating and organizing content. The process of formatting information is automated, which greatly reduces the need for writers to act as their own desktop publishers. However, writers do need to learn to assign useful metadata tagging to documents.

Improving compliance with required document structure

United States government contractors, especially those who work with the military, have long been required to deliver documents using specific standards. The aerospace industry also has specific rules for documents such as aircraft maintenance manuals, and the pharmaceutical industry has rules for labeling. SGML and XML have been used heavily to enforce such standards.

Does your organization need structure?

Armed with basic information about structured authoring, the next logical question is whether your publishing workflow should be moved to a structured environment. In some scenarios, the decision is simple:

- *Content interchange.* XML provides an excellent medium for content interchange. If you need to move content from one format to another, structured content will allow you to automate and systematize the process.

- *Enforcing uniformity across a document set.* Defining a structure lets you apply and enforce consistency across documents. Larger workgroups, higher turnover, and complex formatting requirements for output all make the automation provided by a structured workflow more appealing.

- *Content management.* XML files are in text format, which lends itself to setting up a repository for storage. You can also divide files into small chunks and place them in the repository. The larger the volume of content being produced, the more useful and compelling content management becomes. Structure is not the solution for all content development workflows. In some environments, implementing structure will be more trouble than it's worth. The following are some examples where structure probably doesn't make sense:

- *Fiction and other creative writing.* Fiction is unlikely to fit into a predefined structure, and it probably doesn't require the type of reuse and management that technical content does.

- *Low-value content.* If you do not plan to reuse content, or if a document doesn't contain sufficient information, the effort of structuring it is probably not worth it. Day-to-day business communications, such as email and memos, generally fall in this category. Be on the lookout, though, for higher-value content, such as complex proposals, that could be reused.

- *Small sets of technical content.* Organizations with thousands of pages of content need to consider structure. Organizations with tens of thousands or more pages almost certainly

need both structure and content management. An organization that only manages 100 pages of content doesn't need elaborate structure and content management. Somewhere between 100 and 1,000 pages, there is a point where the value of structure outweighs the implementation cost.

Implementing a structured workflow

If you decide to establish a structured workflow, expect a lengthy and probably painful transition. In an environment where formatting templates are already established and enforced consistently, the addition of structured templates should be relatively straightforward. A workgroup making a transition from a "free-form" authoring environment where templates aren't used to structured authoring should expect major disruption. Structured authoring will completely change the authoring experience.

A minimal implementation process requires that you do all of the following:

- Analyze content and develop structure definitions

- Design a new publishing workflow

- Roll out the new workflow

- Train users

- Set up a maintenance process

Analyzing content and developing structure definitions

Document analysis requires different skills from template design. Instead of creating formatting tags based on a document's appearance, the document architect must identify content elements. Often, formatting is a visual indicator of structure (for example, headings are usually larger than surrounding text), but structure elements may be needed in areas where formatting does not provide a cue.

The document architect begins by reviewing existing documents and analyzing their structure. Any structure that's developed must also take into account new document types that might be needed.

Analysis should also include consideration of how well content meshes with industry-standard structures. Adopting a standard means significantly less development time because DTDs and schemas are often available at no cost.

> **Note:** Some established standards are DITA for topic-based technical documentation (http://dita.xml.org/), S1000D for military equipment (http://www.s1000d.org), and SPL for pharmaceutical labeling (http://www.fda.gov/oc/datacouncil/spl.html).

Designing a new publishing workflow

Once a structure definition is established, it's time for the most controversial part of the implementation process—choosing tools. The following tools will be needed:

- Authoring tool for creating structured documents

- Formatting tool in which automated formatting definitions are set up
- Content management system to keep track of content

A detailed discussion of tools is beyond the scope of this document. The content management system is likely to be by far the most expensive component of a structured workflow and requires the most extensive analysis.

Rolling out the new workflow

A rollout will require two major tasks: notifying users about what's coming and installing the software, servers, and systems that make everything work. Larger numbers of users will add complexity, as will different location types. For example, rolling out a new system to users in two offices would be relatively simple. Integrating hundreds of users in remote home offices adds a degree of difficulty.

Training users

Users need training in several different knowledge areas:

- Structured authoring concepts
- Basic XML concepts
- Creating usable metadata
- Working with a content management system

If writers are not accustomed to creating content for multiple output formats, they may also need training on how to write modular, delivery-neutral information.

Setting up a maintenance process

Once the structured workflow is established, it's critical to set up a process that allows authors to request changes to the structure and the metadata framework.

Summary

Structured authoring offers the prospect of automated formatting and better management of information. New skills are required both to implement a structured workflow and to work within it. Treating content as complex data that can be managed and manipulated requires a significant shift in mindset from authors, editors, and other publishing professionals.

About the authors

Sarah O'Keefe is the chocoholic founder and CEO of Scriptorium Publishing and a content strategy consultant.

Scriptorium offers content strategy services to help customers solve business problems and achieve strategic goals. Since founding the organization in 1997, Sarah has worked with numerous organizations to assess content issues, develop solutions, and implement new technologies that unlock content value. She is the coauthor of Content Strategy 101: Transform Technical Content into a Business Asset.

An experienced public speaker, Sarah is in demand at conferences worldwide. Sarah is fluent in German, reads voraciously, and enjoys swimming, kayaking, and other water sports along with knitting and college basketball. She has strong aversions to raw tomatoes, eggplant, and checked baggage.

You can follow Sarah on Twitter at @sarahokeefe.

Alan Pringle, Chief Operating Officer, helps clients solve problems with the development and distribution of content. He has been with Scriptorium since its inception. His responsibilities include content strategy analysis, automating production and localization tasks (often with XML-based workflows), and managing schedules and budgets for complex consulting projects.

He is coauthor of multiple books, including *Content Strategy 101*. You can follow Alan on Twitter at @alanpringle.