8

ABSTRACTIONS IN THE LwDITA CONTENT LIFECYCLE

The scenario is pretty common in professional conferences: an instructor or trainer attends an event and is inspired by a presentation or visit to a vendor's booth that focuses on a new software application for automating content processes. The instructor requests a free demo or obtains departmental support to purchase an academic or trial license. The curriculum in the instructor's program is modified to include the selected tool, and then the tool developer releases a paid upgrade, which the instructor's department cannot afford. The result: rapid obsolescence and a failed experiment that keeps the department from exploring technologies for creating intelligent content.

Saul Carliner sees these choices of time and money in tools as a risky investment for practitioners and academics alike. For practitioners, "the primary issue is which technologies they should choose for investing their limited training dollars," based on the rapid changes in software applications. For academics, the risks include both "limited resources to purchase costly publishing software" (including enterprise-level content management systems), and the perishable nature of tools-based training, "often outdated within five years," for students (Carliner, 2010, p. 47).

Instead of purchasing a commercial software application and then building a curriculum around it, technical communication instructors and trainers can focus on the mental activities that lead to the development of intelligent content. Those activities are an implementation of the "broader rhetorical strategies for analyzing the audience, planning, revising, and managing one's time and writing process" proposed in Linda Flower's "Rhetorical Problem Solving: Cognition and Professional Writing" (1989, p. 5).

The thinking and working process that I propose in this book involves a series of layers of abstraction based on the model introduced at the end of Chapter 7. For the purpose of this chapter, a revised version of those layers, focusing on the flexibility

about authoring formats allowed by LwDITA and inspired by the ISTE standards for evaluating students as computational thinkers (ISTE, 2018), looks as follows:

- Layer 1: Developing a content strategy
- Layer 2: Authoring modular content with LwDITA
- Layer 3: Separating content from design and context
- Layer 4: Linking topics and maps for collection and reuse
- Layer 5: Processing and producing deliverables
- Layer 6: Preparing for the future.

The "Learning to Think Like a Computer" article from The New York Times mentioned in Chapter 7 provides an appropriate example for looking at each layer of abstraction and its connections to the ISTE standards and the content-development lifecycles presented in this chapter. Dan Garcia's milkshake recipe from that article can be the inspiration for a fictional content project that can guide us through these revised layers of abstraction.

As I focus on each layer of abstraction, I attempt to connect it to stages in content-development lifecycles and the ISTE standards for evaluating computational thinkers. My objective, as I mentioned before, is to separate the intimidating mandate of learning about industry practices and then bringing them back to the classroom in manageable tasks that stay relevant to our profession and also relate to computing-for-all initiatives in the disciplinary context of content development and technical communication, as an alternative to learning about code and programming in introductory computer science courses (although I strongly believe that every student, regardless of academic major, can benefit from an introductory course in computer programming). The correspondence among these layers of abstraction and the indicators in the ISTE standard has not been evaluated empirically beyond my own assessment of students' writing and their progression towards completing the undergraduate degree in Professional and Technical Writing at Virginia Tech.

Layer 1: Developing a Content Strategy

Few concepts have such a strong potential for uniting and dividing, at the same time, opinions about the importance of planning for the development and management of content as *content strategy*. Clark (2016) notes that the idea of content strategy "can realize the interdepartmental possibilities" involving the adoption of reuse and single-sourcing outside of technical communication that authors have been promising for decades. However, Clark also analyzes the discrepancies in definition and implementation of content strategy in the realms of documentation, marketing, web development, and others. Clark's meticulous literature review on the term is eye-opening and inspiring to take the first step through the layers of abstraction behind developing intelligent content with

LwDITA. Authors must be guided by a content strategy, and its development and adoption processes require mental abstractions to defer further layers of the authoring lifecycle until a plan is ready to implement. Without a content strategy, a user cannot advance in the LwDITA process layers of abstraction. Andersen debunks the role of technology over strategy when she explains that while "high-speed networks and the evolution of content technologies (e.g., Web 2.0, CM systems) and their underlying standards (e.g., XML, HTML5, DITA) have made possible the emergence of intelligent content (. . .), its success depends on the content strategy that governs its life cycle" (2014, p. 133).

A simple and direct definition of content strategy, for those who do not have time to read Clark's excellent analysis of the term, comes from Rahel Anne Bailie, who is an active member of the Lightweight DITA subcommittee at OASIS. Bailie defines content strategy as "the analysis and planning to develop a repeatable system that governs the management of content throughout the entire content lifecycle" (2014, p. 14). Going back to the adaptive milkshake recipe from Dan Garcia's example, in a LwDITA workflow an author would need to (following Bailie's definition) analyze and plan before writing any content. The abundance of definitions and approaches to content strategy also creates a long list of potential deliverables related to this concept.

Let's assume that the overarching task is to develop a collection of soda fountain recipes and procedures. For the milkshake entry, the collection should include a versatile recipe with a list of ingredients and steps. The ingredients list should accommodate different flavors in one single template, instead of having separate recipes for each possible flavor. For the scope of this chapter, this layer is not a comprehensive collection of content strategy materials; instead, it focuses on the mental tasks that, at a minimum, authors should focus on before starting an intelligent content project with LwDITA.

1. Perform an audience analysis. Determine who will be the users of the milkshake recipe. If possible, conduct observations and interviews. Bailie & Urbina recommend developing personas and scenarios to understand the audience. Personas, they claim, "help you understand the behavioral characteristics of typical content consumers. From that information, it's possible to anticipate what the most common tasks will be and what information will be in highest demand" (2013, p. 20). Getto & St. Amant (2014) present an overview of personas and their usage at the intersection of user experience and technical communication. Through personas and scenarios, the author can identify the users' needs, expectations, and anticipated reactions to the recipe/collection of recipes. Then, those needs statements can be converted into specific topics and tasks for the recipes.
2. Conduct a content audit or inventory. The author could be creating original content for the recipes or collecting legacy documentation (in this case, as Dan Garcia mentioned, from the pages of an existing cookbook). If the

milkshake represents a truly revolutionary approach to frozen treats, the author should work with the soda fountain experts to document their ideas and tasks while identifying all sources of content. Traditionally, a content audit is based on a spreadsheet with columns representing bits of information and rows for each piece of content. Halvorson & Rach (2012) say that there "is no one perfect format, size, or timing for an audit; there are many different (and totally valid) ways to audit your content," and they recommend including at least the following columns:

- ID: Each content unit needs an identification number or code. A content unit can include a whole milkshake or root beer float recipe, but a picture illustrating how to use the mixer and a how-to video for making whipped cream are also content units that need their own row and ID.
- Title/Topics: Most recipes will have a clear title; other smaller units can use "a short description of the key topics or themes covered."
- URL: If the content units live on the web.
- Format: Explain if the content unit is a piece of "text, video, PDF, etc."
- Source: "Specify whether the content is created in-house, by a content partner (newsfeeds, articles, blog posts, and so on), or by your users."
- Technical home: The content units reside in computer files that are stored somewhere. If the soda fountain has several folders for recipes and images, the "home" for each unit should be specified to avoid missing pieces and duplication.
- Metadata: Lauren Creekmore defines metadata as "attributes of content you can use to structure, semantically define, and target content" (in Abel & Bailie, 2014, p. 28). These attributes are foundational for the filtering and processing of content that will take place in future layers. An example would be a context metadata for the milkshake recipe. The recipe could be the same, with some variations about available equipment and ingredients, for personnel preparing it in a restaurant or a food truck environment. The metadata, which end users do not see, can set specific rules for each of those environments. After processing, the resulting recipe will give the exact instructions for each context. Layer 3 provides more details about this sample use of metadata.
- Traffic/usage statistics: "If it's feasible, get the skinny on how people are using (or not using) each piece of content."
- Last update: "When was the last time somebody in your organization paid attention to this piece of content?"
- Language: "If you have content in multiple languages, you'll want to record the language or dialect used on each piece of content" (Halvorson & Rach, 2012, pp. 51–52).

3. Develop a content structure. Structured authoring can provide many benefits, including consistency of format and sections among recipes. The

recipe can be its own content type specific characteristics and headings. Sara Wachter-Boettcher (2012) actually includes the recipe in her list of common examples of content types, which also includes bios, blog posts, business listings, episodes, event listings, fact sheets, FAQs, feature articles, help/user assistance modules, podcasts, poems, press releases, products, reviews, short stories, testimonials, tips and lists, and tutorials. In Chapter 1 we saw how DITA XML is frequently associated in the world of technical communication with the content types of concept, task, and reference. Chapters 5 and 6 guided us through the role of content types in LwDITA, but for the purpose of the milkshake recipe we can think of a structured topic that includes sections for ingredients and tools, steps, and a result.

4. Compile or adopt a style guide and code of ethics. If the collection of recipes has more than one author, a style guide can keep all recipes consistent in punctuation, capitalization, word usage, and other editorial decisions. Adopting an existing style guide would be an easy decision, but maybe the owner has specific ideas about content formatting. Similarly, adopting a code of ethics can avoid leaving users at a disadvantage by establishing principles. Russell Willerton (2015) assembled a thorough overview of ethics in the technical and professional communication literature that includes a section on characteristics of available codes for the profession.
5. Create a diversity plan. Authors should acknowledge the diversity of their content users, which can include local issues such as including information for milkshakes that use non-dairy alternatives instead of milk. The diversity plan should also consider outlining a global content strategy, which Val Swisher defines as "a plan for managing content that is intended for people whose main language is something other than the source language" (2014). The plan should also acknowledge the diversity of content *authors*, which includes language but also covers the diverse departments or groups involved with the content. If the soda fountain project were to expand, authors could include professionals from the kitchen, technical writers in charge of instructions, and marketing writers developing promotional materials. Those groups have different communication styles, expectations, and structures that should be acknowledged and reconciled.
6. Make a technology plan considering human involvement. Following from the previous task, this one should identify the writing platforms, content standards, and tools used by the potential authors. The technology plan should also cover decisions about content management applications or services, and tools for processing of deliverables. The plan should also consider and accommodate a human-in-the-loop (HITL) strategy. Rothrock & Narayanan posit that traditional projects involving automation "regard human interaction as an external input to the system being considered. However, studies of complex systems in today's technological landscape must include humans as active participants" (2011, p. v). For this example,

the plan includes humans as content developers and curators working with a user experience team to produce end-user deliverables for the soda fountain. For the milkshake example, the selected standard will be LwDITA, authored in the following two options:

- Option a: With the text editor Atom by GitHub, with the DITA Open Toolkit as processing tool, and with content and source code stored and delivered in GitHub and GitHub Pages, respectively.
- Option b: With the LwDITA-aware application Oxygen XML, and with content and source code stored and delivered in GitHub and GitHub Pages, respectively.

The tasks involved in this layer are related to several stages of the content-development lifecycles featured in the previous chapter. Layer 1 tasks focus on Bailie & Urbina's *analysis* stage. They also cover some of the work included in Andersen's *Analyzing the customer and business needs* and *Developing an information architecture* stages. Contrasting layer 1 with the ISTE standards for assessing computational thinking in students, some of its tasks could comply with sections of standards 5a (*Students formulate problem definitions suited for technology-assisted methods such as data analysis, abstract models and algorithmic thinking in exploring and finding solutions*), 5b (*Students collect data or identify relevant data sets, use digital tools to analyze them, and represent data in various ways to facilitate problem-solving and decision-making*), and 5c (*Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving*).

Layer 2: Authoring Modular Content with LwDITA

Probably the most important layer for readers of *Creating Intelligent Content with Lightweight DITA*, layer 2 involves content development tasks that can only happen after completing the first layer of abstraction. Following the model proposed by Flower (1989), this layer focuses on important discourse conventions that, regardless of their inherent importance, need the broader rhetorical conventions of the remaining layers. Once a content audit has identified all legacy documentation, and it has been prepared for update or migration, it's time for authors to produce text and/or multimedia content to address the users' needs identified in layer 1. For the milkshake example, let's assume that the soda fountain operator manual will include a few recipes, with the versatile milkshake being one of them, probably featured next to the root beer float and the banana split. The authors, after collecting information from any existing sources, conducting interviews with experts, and observing work in the kitchen, can start creating topics following the guidelines from the proposed LwDITA standard.

Focusing, for now, exclusively on the milkshake recipe and ignoring all other offerings from the fountain manual, we can identify more than one way to achieve with LwDITA the versatility required by Dan Garcia's example. One possible approach could be through *conditional content*, which Julio Vazquez defines as "content that has sufficient metadata to allow a processor to filter or flag that content in any output or format, using a profile to determine the exact output for a given context or format" (2016, p. 64). Layer 3 will look at conditional content, as we advance to connecting elements of the recipe to specific contexts. A second approach, which is more appropriate for layer 2, is provided by *content variables*, which Nancy Harrison defines as "variables that contain phrase-level content that needs to be in a topic no matter what document the topic is part of, but that changes depending on context, for example, a product name or a company name" (2016, p. 66).

The content variable *"icecream-flavor"* appears in a phrase-level element of the milkshake recipe (inside an ingredient item), but its value changes depending on the flavor established by the manager. If the milkshake today will be strawberry-flavored, the variable *"icecream-flavor"* will be set to the value of *"strawberry"*. If tomorrow the flavor needs to be vanilla, the variable can take that new value and keep the same content with only that variation. That variable should be part of the metadata specified in the previous layer, as the author identified elements that need to change in the recipes. Without that metadata, an author cannot produce the adaptive milkshake recipe required by Dan Garcia's example. The following sections show how to use a content variable in the three initial authoring formats of LwDITA.

XDITA

In XDITA, the LwDITA authoring format based on XML, a content variable is implemented on the phrase (<**ph**>) element using the **@keyref** attribute. Hackos explains the **@keyref** mechanism in DITA XML, as a process of "putting placeholders into your topics and then defining the content for those placeholders" elsewhere (2011, p. 272). The **@keyref** mechanism, Hackos adds, is comprised of two components:

- The referencing key (the **@keyref** attribute), which is "found in the topic where the content will be included," and
- The defining key (the **@keys** attribute), which uses the <**keydef**> element to "set the content to replace the placeholder" (Hackos, 2011, p. 273).

The referencing key is covered in this layer, and the defining key will be set in layer 4. Layer 5 addresses the processing tasks that will populate the content placeholders with the value established in the defining keys.

In the versatile milkshake recipe authored in XDITA, the referencing key will be inside the ingredient that specifies a pint of ice cream. The content placeholder will leave the ice cream flavor initially empty, and then it will adopt a value determined by the defining key in layer 4. Figure 8.1 shows a version of the XDITA code for the milkshake recipe. Like all code samples from previous chapters, the examples included in this section can be authored in any text editor (not in a word processor) or a LwDITA-aware software application, and can be downloaded from the *Creating Intelligent Content with Lightweight DITA* GitHub repository (https://github.com/carlosevia/lwdita-book).

The code from Figure 8.1 (let's call it a draft of the recipe) includes an unordered list (**<ul>**) with the identifier of "ingredients." This list of ingredients contains two list item (**<li>**) elements. The first one is straightforward text that specifies some milk (a more advanced recipe could even specify if the fountain attendant should use actual milk or a non-dairy milk alternative). An author working with this source code would be able to tell that step 1 is asking for ¼ cup of milk[1]. Step 1 does not require any abstraction beyond understanding text and the English language. The second list item, however, has an XDITA phrase (**<ph>**) element inside the list item. This phrase element has a **@keyref** attribute that works as a placeholder for variable text. The **@keyref** attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD LIGHTWEIGHT DITA Topic//EN"
"lw-topic.dtd">
<topic id="milkshake">
 <title>Easy Milkshake</title>
 <body>
    <p> The Easy Milkshake is our best-seller and a soda fountain tradition. We frequently
update the recipe to incorporate fresh flavors and ingredients.</p>
<ul id="ingredients">
    <li><p>1/4 cup of milk</p></li>
    <li><p>A pint of <ph keyref="icecream-flavor"/> ice cream</p></li>
  </ul>
  <ol id="steps">
<li><p>Combine all ingredients in the Blendimixx 3000</p></li>
<li><p>Mix for 30 seconds</p></li>
<li><p>Serve in a cold fountain glass</p></li>
  </ol>
</body>
</topic>
```

FIGURE 8.1 Recipe for a versatile milkshake in XDITA – following on an idea presented by Dan Garcia. In this example, the recipe was created in XDITA – the LwDITA authoring format based on a simplified version of DITA XML. The list item asking for a pint of ice cream includes a key reference with the value of *"icecream-flavor,"* which will be set at processing time in a different layer.

has the generic value of *"icecream-flavor"* . Ingredient 2, by itself, would be very confusing for a reader expecting a milkshake recipe. The **@keyref** attribute has to be established in this authoring layer, but it depends on the defining key to be specified in layer 4. As a result, the LwDITA topic for the milkshake recipe is not ready for processing in this layer. The author can save this recipe as *milkshake.dita*, but even a LwDITA-aware editor or the DITA Open Toolkit would indicate an error if the author attempts to build a deliverable at this stage.

HDITA

The same milkshake recipe can be created in HDITA, the LwDITA authoring format based on HTML5, with a similar mechanism for the content variable. In HDITA, the referencing key for variable content is expressed with the custom data attribute **@data-keyref**. In HTML5, phrase-level content can be included in the **** element. Thus, the recipe in HDITA would look like the draft in Figure 8.2.

The unordered list element with the identifier of "ingredients" is in this version and it looks very similar to the one included in the XDITA recipe. The main

```
<!DOCTYPE html>
<title>Easy Milkshake</title>
<body>
<article id="milkshake">
 <h1>Easy Milkshake</h1>
 <p>The Easy Milkshake is our best-seller and a soda fountain tradition. We frequently update
the recipe to incorporate fresh flavors and ingredients.</p>
<ul id="ingredients">
   <li>
    <p>1/4 cup of milk</p>
   </li>
   <li>
    <p>A pint of <span data-keyref="icecream-flavor"></span> ice cream</p>
   </li>
</ul>
 <ol id="steps">
<li><p>Combine all ingredients in the Blendimixx 3000</p></li   >
<li><p>Mix for 30 seconds</p></li>
<li><p>Serve in a cold fountain glass</p></li>
 </ol>
</article>
</body>
```

FIGURE 8.2 Recipe for the versatile milkshake authored in HDITA. The ingredient asking for a pint of ice cream uses the HTML5 custom data attribute @data-keyref to set a placeholder for a value that will be processed in a more advanced layer.

differences are in the element holding the variable content (**** in HDITA, whereas in XDITA it was **<ph>**) and the attribute setting the placeholder *"icecream-flavor"* for a flavor to be determined in a more advanced layer (**@data-keyref** in HDITA, whereas in XDITA it was **@keyref**). These differences keep the LwDITA authoring formats compliant with the HTML5 and DITA XML standards, respectively. The author can save this recipe as *milkshake.html*, and the resulting file can be viewed in a web browser, but the recipe won't make sense for end users because the content variable for "icecream-flavor" is not established in this layer of abstraction (Figure 8.3).

MDITA

The versatile milkshake recipe can be expressed, with some minor differences, in the MDITA core profile. MDITA uses the shortcut reference

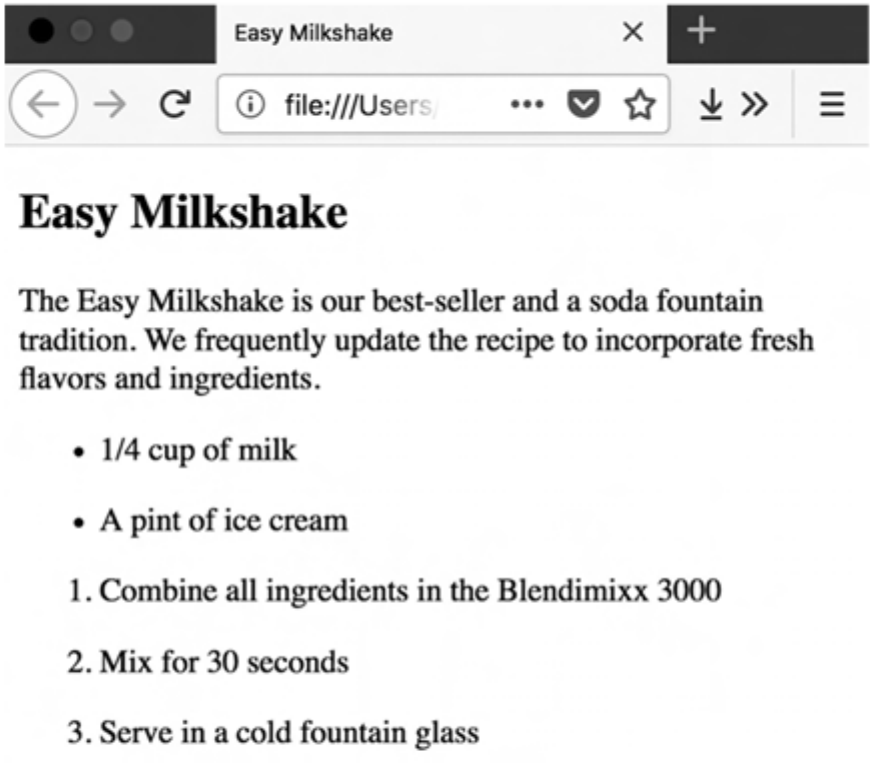


FIGURE 8.3 HDITA version of the milkshake recipe as seen on a web browser. Note that the ingredient for a pint of ice cream does not have a flavor attached to it, which will come in a more advanced layer when the topic is processed through a LwDITA-aware software application.

link structure from CommonMark/GitHub Flavored Markdown (GFM) to establish content variables. The GFM spec explains shortcut reference links as consisting "of a link label that matches a link reference definition elsewhere in the document and is not followed by [] or a link label. The contents of the first link label are parsed as inlines, which are used as the link's text. The link's URI and title are provided by the matching link reference definition. Thus, [foo] is equivalent to [foo][]" (GitHub Flavored Markdown Spec, 2017).

The milkshake recipe authored in MDITA extended profile, with a GFM shortcut reference link setting the content placeholder for *"icecream-flavor"* , would look like Figure 8.4.

The unordered list item for ingredients is still in the recipe, and the second ingredient has the placeholder variable *"icecream-flavor"* , which will inherit a value in a more advanced layer of abstraction. The author can save this MDITA topic as *milkshake.md*.

The tasks involved in this layer are related to stages of the content-development lifecycles featured earlier in the previous chapter. Layer 2 tasks focus on Bailie & Urbina's *collect* stage. They also cover some of the work included in Andersen's *Creating structured content* stage. Contrasting layer 2 with the ISTE standards for assessing computational thinking in students, some of its tasks could comply with sections of standards 5a (*Students formulate problem definitions suited for technology-assisted methods such as data analysis, abstract models and algorithmic thinking in exploring and finding solutions*) and 5b (*Students collect data or identify relevant data sets, use digital tools to analyze them, and represent data in various ways to facilitate problem-solving and decision-making*).

```
# Easy Milkshake

The Easy Milkshake is our best -seller and a soda fountain tradition. We frequently update the
recipe to incorporate fresh flavors and ingredients.

- 1/4 cup of milk
-A pint of [icecream-flavor] ice cream

1.  Combine all ingredients in the Blendimixx 3000
2.  Mix for 30 seconds
3.  Serve in a cold fountain glass
```

FIGURE 8.4 Recipe for the versatile milkshake authored in MDITA. The ingredient for ice cream uses a GitHub Flavored Markdown shortcut reference link to inherit a key reference from a map that will appear in a more advanced layer.

Layer 3: Separating Content from Presentation and Context

In a small soda fountain, the operators probably will have a printed manual as the only available documentation. In a larger operation, however, the content of a recipe collection could be used (or single-sourced) in the fountain's operation manual, website, social media presence, mobile app, and even conversational guidelines for a device like the Amazon Echo. In the first scenario, it makes sense to have an author in control of the manual's content and presentation to save time and money: one deliverable, one format, and one author. In the second scenario, however, an intelligent content solution to address all those potential deliverables depends on separating content from presentation. For years, I have taught in my courses the four levels of Pringle & O'Keefe's methodology for developing technical documents:

- Chaos ("there's no consistency in the presentation of content")
- Page consistency ("content looks the same on paper (or other delivery format," but there's no consistency in its source files)
- Template-based authoring (content follows "predetermined styles (and) writers don't spend time figuring out how to create particular formatting — they apply styles to add formatting")
- Structured authoring ("a publishing workflow that defines and enforces consistent organization of content") (2009, pp. 41–42).

In previous layers of abstraction, the milkshake recipe moved towards the structured authoring level. As it prepares for more advanced layers, the recipe must keep content and presentation separate. This separation "can create philosophical and cognitive dissonance for technical communicators trained to think of information as content that is inherently linked to presentation" (Clark, 2007, p. 36). Mark Baker warned about the difficulties of teaching this layer in the following statement:

> One of the hardest things about moving technical writers from desktop publishing to structured writing is persuading them to give up responsibility for how the final output looks. Writers will keep looking for ways to specify layout, even in markup languages specifically designed to remove layout concerns. They understand their jobs in terms of the responsibilities their old tools imposed on them.
>
> (Baker, 2013, p. 87)

The presentation rules and details for the milkshake recipe will be provided in a further layer of abstraction by an external tool and style sheet. After completing layer 2, the recipe topics look either like raw XML, HTML5, or Markdown

code, depending on the LwDITA authoring format used to create them. In this layer, authors should focus on producing content (text, audio, videos, etc.) and letting the presentation be addressed elsewhere.

The fear of losing control over context is, however, real. According to some, writers separating content from presentation "will have no control over the context in which their information appears or the uses to which it may be put" (Gu & Pullman, 2009, p. 6). Those concerns echo a threat to the characteristic of rhetorical effectiveness, and DITA has tried to ameliorate that effect by giving authors control on the context of their content elements. Albers called for a similar dimension of rhetorical effectiveness when discussing the effects of single sourcing and XML in the careers of technical communicators: the real questions in documentation projects "should not revolve around technology but around whether the resulting documents effectively address a user's real-world information needs (Albers, 2003, p. 338). Swarts also expressed a need for content management systems that "suggest a rhetorical use of the content that writers have at their disposal" (Swarts, 2010, p. 159).

In a workflow like the one proposed in this chapter, authors and their supervisors take care of planning, authoring, organizing, and evaluating activities to prevent content-generation problems. When dealing with machine automation, there will always be the possibility of errors and the occasional misplaced component in a document, but those are more the exception than the rule. The human-in-the-loop strategy developed in layer 1 has the goal of refuting portrayals of writers as the lonely humans in a machine-dominated process of content automation, with the content products of their hard work being *arhetorically* assembled (borrowing a term from Bacha, 2009) by algorithms and machines without human control. The metadata and rules included in topics, phrases, and maps preserve the rhetorical effectiveness of intelligent content repositories.

LwDITA inherits the context-setting capability of DITA XML, which depends on rich metadata to filter or flag content. This type of context functions like a simple conditional statement in the "if-then" model of computer programming.

Taking this to our recipe example, let's give the soda fountain an official downtown location and also a mobile location with a food truck. Most processes are the same in both locations, but some change because of available equipment and ingredients. In the downtown location, the soda fountain operators have a professional *Blendimixx 3000* mixer, which they use for the milkshakes; however, in the food truck location they have a smaller *Blendilitte 200*. The sample recipe from layer 2 only mentions the professional mixer, but now that its content will also generate the operator manual for the food truck location, this same recipe should accommodate both cases while minimizing distractions for the users: in the downtown location, the manual should only mention the professional mixer, but in the food truck location it should only mention the smaller blender. The metadata attributes assigned in this layer will help filter out irrelevant information for each context in a future layer of abstraction.

DITA XML has several attributes that identify properties for conditional processing, which include **@audience**, **@plaftorm**, and **@product**. In LwDITA, the only conditional processing attributes are **@props** (in XDITA) or **@data-props** (in HDITA and MDITA extended profile). The DITA 1.3 spec defines **@props** (properties) as a "generic conditional processing attribute", and in LwDITA it can take different values based on authors' needs (2.2.4.2.1 Conditional processing attributes, 2018).

In XDITA, the recipe will look like Figure 8.5 with the added metadata for context,

The XDITA recipe, which can be saved as *milkshake.dita*, has a list item on the ordered list labeled "steps" that instructs the operator to combine all ingredients in a blender. The topic shows both options (the *Blendimixx 3000* and the *Blendilitte 200*), which have attached the corresponding metadata for each context with the conditional attribute **@props** inside a phrase element. In this layer of abstraction, the XDITA topic displays all available options for equipment and locations. The topic is written in valid XDITA (it conforms to the rules of the LwDITA standard) and can be processed in a software application to generate user deliverables. However, those user deliverables will be incorrect or redundant because the filters for specific context will be applied in a future layer. Figure 8.6 shows a PDF transformation of the XDITA milkshake recipe as it looks in layer 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD LIGHTWEIGHT DITA Topic//EN"
"lw-topic.dtd">
<topic id="milkshake">
 <title>Easy Milkshake</title>
 <body>
    <p>The Easy Milkshake is our best-seller and a soda fountain tradition. We frequently
update the recipe to incorporate fresh flavors and ingredients.</p>
<ul id="ingredients">
    <li><p>1/4 cup of milk</p></li>
    <li><p>A pint of <ph keyref="icecream-flavor"/> ice cream</p></li>
  </ul>

  <ol id="steps">
<li><p>Combine all ingredients in the <ph props="setting-downtown">Blendimixx
3000</ph> <ph props="setting-foodtruck">Blendilitte 200</ph></p></li>
<li><p>Mix for 30 seconds</p></li>
<li><p>Serve in a cold fountain glass</p></li>
  </ol>
</body>
</topic>
```

FIGURE 8.5 XDITA version of the milkshake recipe with two possible values for a setting condition. At processing time, the author can specify a filter and exclude the value that is not needed for a specific setting.

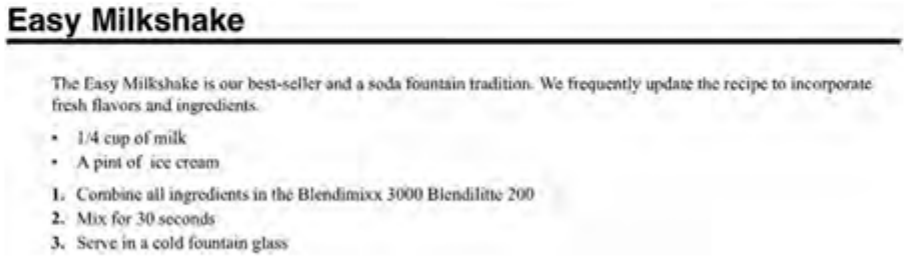


FIGURE 8.6 PDF transformation of the XDITA milkshake recipe. Since no filters were applied during the transformation, the first step includes the values of both blenders without considering the setting where the recipe will be used. Also, there is no ice cream flavor set, as the content variable will be populated in a more advanced layer.

In HDITA, the conditional attribute **@data-props**, placed inside the HTML5 **** element, can achieve similar results. An HDITA version of the milkshake recipe, with context information to specify the blender used in each location, would look like Figure 8.7.

The HDITA recipe, which can be saved as *milkshake.html*, is a valid HTML5 topic; therefore, it can be seen on a web browser at this stage. The resulting web

```
<!DOCTYPE html>
<title>Easy Milkshake</title>
<body>
<article id="milkshake">
 <h1>Easy Milkshake</h1>
 <p>The Easy Milkshake is our best-seller and a soda fountain tradition. We frequently update
the recipe to incorporate fresh flavors and ingredients.</p>
<ul id="ingredients">
  <li>
   <p>1/4 cup of milk</p>
  </li>
  <li>
   <p>A pint of <span data-keyref="icecream-flavor"></span> ice cream</p>
  </li>
</ul>
 <ol id="steps">
<li><p>Combine all ingredients in the <span data-props="setting-downtown">Blendimixx
3000</span> <span data-props="setting-foodtruck">Blendilitte 200</span></p></li>
<li><p>Mix for 30 seconds</p></li>
<li><p>Serve in a cold fountain glass</p></li>
 </ol>
</article>
</body>
```

FIGURE 8.7 HDITA version of the milkshake recipe with two possible values for a setting condition, which are specified with the HTML5 custom data attribute **@data-props**.

rendering (Figure 8. 8), however, shows both possible blender options in the same step and would be confusing for an actual human operator. In layer 5, processing the topic with the proper conditional code will produce correct deliverables for each location.

The context-aware recipe cannot be expressed in MDITA core profile. In order to use the conditional variable **@data-props**, a Markdown-based author would have to use a few code snippets of HDITA in MDITA extended profile. The recipe main elements are still in Markdown, but the available values for the conditional setting attribute are the same as those used in the HDITA topic, as shown in Figure 8.9. This MDITA version can be saved as *milkshake.md*.

The HDITA code snippets enable the MDITA topic to allow conditional content and let Markdown authors use DITA-like content filters. The MDITA topic, in this layer, would not make sense for human users, and it must be processed by a LwDITA-aware tool in layer 5.

The tasks involved in this layer are related to several stages of the content-development lifecycles featured earlier in the previous chapter. Layer 3 tasks focus on Bailie & Urbina's *manage* stage. They also cover some of the work

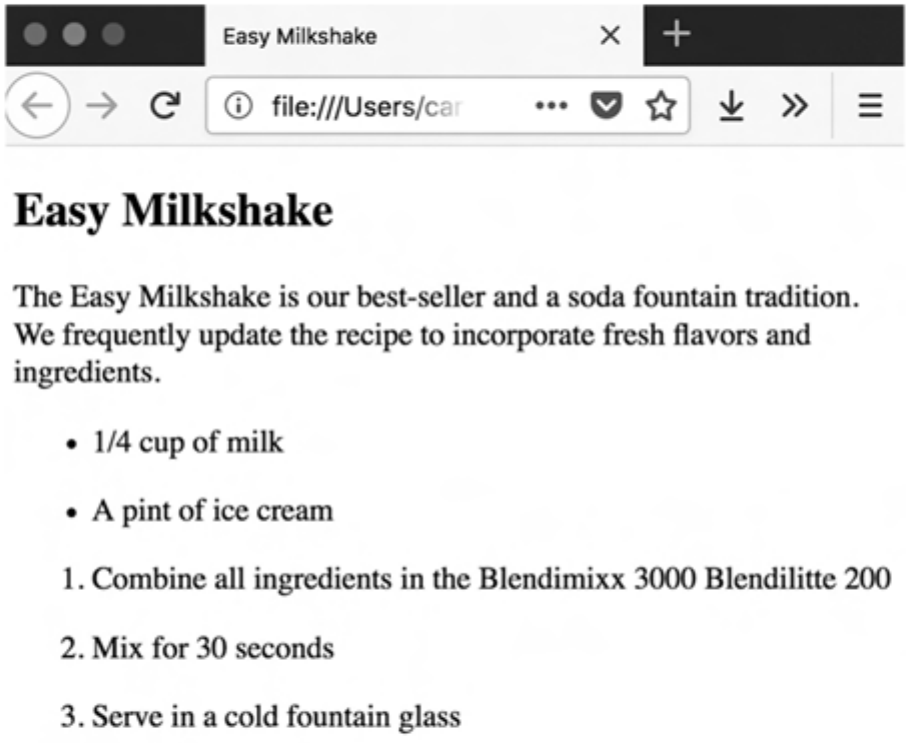


FIGURE 8.8 HDITA version of the milkshake recipe seen through a web browser. Without advanced processing, the HTML file shows the values for both possible blenders in step 1.

```
# Easy Milkshake

The Easy Milkshake is our best-seller and a soda fountain tradition. We frequently update the
recipe to incorporate fresh flavors and ingredients.

-  1/4 cup of milk
-  A pint of [icecream-flavor] ice cream

1.  Combine all ingredients in the <span data-props="setting-downtown">Blendimixx
3000</span> <span data-props="setting-foodtruck">Blendilitte 200</span>
2.  Mix for 30 seconds
3.  Serve in a cold fountain glass
```

FIGURE 8.9 MDITA version of the milkshake recipe with two possible values for a setting condition. To overcome Markdown's limitations as a language for structuring content, the conditional properties are expressed with HDITA code snippets.

included in Andersen's *Developing an information architecture* and *Creating structured content* stages. Contrasting layer 3 with the ISTE standards for assessing computational thinking in students, some of its tasks could comply with sections of standards 5a (*Students formulate problem definitions suited for technology-assisted methods such as data analysis, abstract models and algorithmic thinking in exploring and finding solutions*) and 5c (*Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving*).

Layer 4: Linking Topics and Maps for Collection and Reuse

Linking Topics to Topics

In Chapter 6, we saw how to represent a cross-reference component in the LwDITA authoring formats. With that component, an author can link a topic to an external web resource, a file (an existing diagram in a PDF, for example), or another LwDITA topic. Cross-topic linking works regardless of authoring format: an XDITA topic can link to topics in HDITA and MDITA, and vice versa. However, the cross-reference component in LwDITA also enables one of DITA's strongest content reuse features: the content reference – or *conref* for short.

Layer 2 introduced the key reference attribute as a placeholder for content reuse. The conref is a related attribute that allows an author to bring content from one topic into another (without copying and pasting). The source element from the content reference can be updated and, after reprocessing the topic collection, all target topics that link to it will be automatically updated. Hackos describes the structure of the conref mechanism in the DITA standard as follows:

> The @conref attribute uses a unique @id attribute to identify the content unit you want to use. For example, if you want to use a <note> from one topic into another, you must add an @id attribute to the original note.
>
> (Hackos, 2011, p. 240)

In LwDITA, the structure of a conref should look like the following template: *path-to-file/file-name#topic-id/element-id*

For example, let's pretend that the LwDITA topic with the milkshake recipe includes a note component, with the type of warning, that tells the operator to unplug the blender after each use to prevent a fire hazard. That note should be presented in every recipe produced by the soda fountain. Instead of copying and pasting the note's content, an author can use a conref to link the topics. The milkshake topic, then, becomes the source for the conref, and in XDITA syntax it would look like Figure 8.10.

In HDITA syntax, the revised milkshake recipe would look like Figure 8.11.

The Lightweight DITA subcommittee at OASIS evaluated several approaches for including content references natively in Markdown syntax. However, some of them required changes to many other MDITA components or allowed content exchange exclusively among Markdown files. Therefore, there is no direct way to express a conref in MDITA core profile, and if authors want to take advantage of this reuse mechanism they would need to include a raw HDITA code snippet in an MDITA extended profile topic. Thus, the MDITA version of the milkshake topic with the warning note would look as Figure 8.12.

Keep in mind that all recipes included in the soda fountain manual will need the warning note. An author could copy and paste from the milkshake recipe into all the other recipes from the manual. However, what if the requirements change and a new type of blender does not require to be unplugged but needs to be descaled once a week? The author would need to update the note manually on every topic that includes it. With a conref, however, the only note that needs to be updated is the source in the milkshake topic, and all recipes that link to it will be updated automatically after re-processing the topics and their map (more about that in the next layer).

To continue with the example, let's look at the root beer float recipe, which is one of the many target topics that link to the warning note from the milkshake recipe. Figure 8.13 shows the XDITA version of the root beer float recipe.

Figure 8.14 shows the root beer float recipe in HDITA syntax.

Figure 8.15 shows the root beer float recipe in MDITA extended profile syntax. The content reference will be in a raw HDITA code block.

The **@conref** (in XDITA) or **@data-conref** (in HDITA and MDITA extended profile) attribute is available on the following LwDITA content components:

- Audio
- Definition description

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD LIGHTWEIGHT DITA Topic//EN"
"lw-topic.dtd">
<topic id="milkshake">
 <title>Easy Milkshake</title>
 <body>
  <p>The Easy Milkshake is our best-seller and a soda fountain tradition. We frequently update
the recipe to incorporate fresh flavors and ingredients.</p>
  <ul id="ingredients">
   <li>
    <p>1/4 cup of milk</p>
   </li>
   <li>
    <p>A p  int of
     <ph keyref="icecream-flavor"/>
     ice cream</p>
   </li>
  </ul>
  <ol id="steps">
   <li>
    <p>Combine all ingredients in the
     <ph props="setting-downtown">Blendimixx 3000</ph>
     <ph props="setting-foodtruck">Blendilitte 200</ph>
    </p>
   </li>
   <li>
    <p>Mix for 30 seconds</p>
   </li>
   <li>
    <p>Serve in a cold fountain glass</p>
   </li>
  </ol>
  <note id="unplug" type="warning">
   <p>Unplug the blender after each use to prevent a fire hazard.</p>
  </note>
 </body>
</topic>
```

FIGURE 8.10 Milkshake recipe in XDITA syntax with the note component. The note has a unique **@id** with the value of unplug, and the optional @type for the note is set to "warning".

- Definition list
- Definition list entry
- Definition term
- Footnote
- List item
- Note
- Ordered list
- Paragraph
- Preformatted text
- Section
- Simple table

```
<!DOCTYPE html>
<title>Easy Milkshake</title>
<body>
 <article id="milkshake">
  <h1>Easy Milkshake</h1>
  <p>The Easy Milkshake is our best-seller and a soda fountain tradition. We frequently update
the recipe to incorporate fresh flavors and ingredients.</p>
  <ul id="ingredients">
   <li>
    <p>1/4 cup of milk</p>
   </li>
   <li>
    <p>A pint of <span data-keyref="icecream-flavor"></span> ice cream</p>
   </li>
  </ul>
  <ol id="steps">
   <li>
    <p>Combine all ingredients in the <span data-props="setting-downtown">Blendimixx
3000</span> <span data-props="setting-foodtruck">Blendilitte 200</span></p>
   </li>
   <li>
    <p>Mix for 30 seconds</p>
   </li>
   <li>
    <p>Serve in a cold fountain glass</p>
   </li>
  </ol>
  <div data-class="note" id="unplug" data-type="warning">
   <p>Unplug the blender after each use to prevent a fire hazard.</p>
  </div>
 </article>
</body>
```

FIGURE 8.11 Milkshake recipe in HDITA syntax with the note component. The HTML5 ** ** element includes the **@id** with the value of "unplug" and custom data attributes to indicate that it is a LwDITA note with the type of "warning".

- Simple table entry
- Simple table header
- Simple table row
- Unordered list
- Video

Linking Maps to Topics

In this layer, we will work with maps as the main mechanism to organize topics and create information hierarchies. LwDITA maps are essential for some mental abstractions that take place in layer 4. The sample milkshake topics that have been evolving in previous layers must be collected in a map in order to produce deliverables for their intended human and algorithmic audiences (see

```
# Easy Milkshake

The Easy Milkshake is our best-seller and a soda fountain tradition. We frequently update the
recipe to incorporate fresh flavors and ingredients.


-1/4 cup of milk
-A pint of [icecream-flavor] ice cream


1.  Combine all ingredients in the <span data-props="setting-downtown">Blendimixx
3000</span> <span data-props="setting-foodtruck">Blendilitte 200</span>
2.  Mix for 30 seconds
3.  Serve in a cold fountain glass


<div data-class="note" id="unplug" data-type="warning">
<p>Unplug the blender after each use to prevent a fire hazard.</p>
</div>
```

FIGURE 8.12 Milkshake recipe in MDITA extended profile syntax with the note component. The conref is represented with a raw block of HDITA code.

Gallagher, 2017). In this section, we will look at an XDITA map that includes the milkshake recipe in a collection of topics that also includes recipes for a root beer float and a banana split. This map can be combined with the topics developed in earlier layers and advance to layer 5, where it will be processed by a LwDITA-aware tool and generate information products for the soda fountain operators. The map in the following example not only works as a collection mechanism for topics, it also provides the defining key for the *"icecream-flavor"* content variable created in layer 2. Figure 8.16 shows a sample XDITA map for the Soda Fountain Operator Manual.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD LIGHTWEIGHT DITA Topic//EN"
"lw-topic.dtd">
<topic id="float">
 <title>Root Beer Float</title>
 <body>
  <ul id="ingredients">
   <li><p>20 oz root beer</p></li>
   <li><p>A large scoop of vanilla ice cream</p></li>
  </ul>
 <ol id="steps">
   <li><p>Pour root beer in a large cold glass</p></li>
   <li><p>Scoop ice cream into the glass</p></li>
   <li><p>Serve float as it starts to fizzle</p></li>
  </ol>
  <note conref="milkshake.dita#milkshake/unplug" />
 </body>
</topic>
```

FIGURE 8.13 XDITA version of the root beer float recipe with a content reference. The conref links to the source note in the milkshake recipe. Note the syntax of **filename#topic-id/element-id**.

```
<!DOCTYPE html>
<title>Root Beer Float</title>
<body>
 <article id="float">
  <h1>Root Beer Float</h1>
  <ul id="ingredients">
   <li>
    <p>20 oz root beer</p>
   </li>
   <li>
    <p>A large scoop of vanilla ice cream</p>
   </li>
  </ul>
  <ol id="steps">
   <li>
    <p>Pour root beer in a large cold glass</p>
   </li>
   <li>
    <p>Scoop ice cream into the glass</p>
   </li>
    <li>
     <p>Serve float as it starts to fizzle</p>
   </li>
  </ol>
   <div data-conref="milkshake.dita#milkshake/unplug"></div>
 </article>
</body>
```

FIGURE 8.14 HDITA version of the root beer float recipe with a content reference. The conref links to the source note in the milkshake recipe.

The XDITA map in Figure 8.16 includes references (**<topicref>**) to the three hypothetical recipes in the manual. The author can save this file as *fountain.ditamap*, and in layer 5 a machine processor will parse the map and referenced topics in XDITA, HDITA, and MDITA. Following the discussion about content silos from layer 2, the map on Figure 8.16 includes topics created in the

```
# Root Beer Float

-20 oz of root beer
-A large scoop of vanilla ice cream

1.  Pour root beer in a large cold glass
2.  Scoop ice cream into the glass
3.  Serve float as it starts to fizzle

<div data-conref="milkshake.dita#milkshake/unplug"></div>
```

FIGURE 8.15 MDITA version of the root beer float recipe with a content reference. The conref links to the source note in the milkshake recipe. Authors can use an HDITA code block to express the content reference.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE map PUBLIC "-//OASIS//DTD LIGHTWEIGHT DITA Map//EN" "lw-map.dtd">
<map>
<topicmeta>
 <navtitle>Soda Fountain Operator Manual</navtitle>
</topicmeta>
 <keydef keys="icecream-flavor">
   <topicmeta>
    <linktext>strawberry</linktext>
   </topicmeta>
</keydef>
 <topicref href="xdita-topics/milkshake.dita" format="xdita"/>
 <topicref href="hdita-topics/float.html" format="hdita"/>
 <topicref href="mdita-topics/bananasplit.md" format="mdita"/>
</map>
```

FIGURE 8.16 XDITA map for a soda fountain operator manual. The **<keydef>** environment declares the value for the "icecream-flavor" variable in any linked topics.

different LwDITA authoring formats (see the different values of **@format** in the **<topicref>** entries). This means that, for example, end users will not know what came from MDITA and what came from HDITA: they will just see a unified deliverable.

The XDITA map also has a navigation title (**<navtitle>**) that can specify the main heading for any user products created in the next layer of abstraction. The defining key for the *"icecream-flavor"* content variable is provided in the key definition (**<keydef>**) environment. Any LwDITA topics referenced in this map will inherit that key definition in layer 5, and all elements with a placeholder for *"icecream-flavor"* will display the key of, in this example, strawberry. If vanilla is the flavor of the day tomorrow, the author only needs to change that in the map key reference, and all the topics referenced will be updated automatically when the collection is processed in layer 5.

The tasks involved in this layer are related to several stages of the content-development lifecycles featured earlier in the previous chapter. Layer 4 tasks focus on Bailie & Urbina's *collect* and *manage* stages. They also cover some of the work included in Andersen's *Analyzing the customer and business needs* and *Developing an information architecture* stages. Contrasting layer 4 with the ISTE standards for assessing computational thinking in students, some of its tasks could comply with sections of standards 5b (*Students collect data or identify relevant data sets, use digital tools to analyze them, and represent data in various ways to facilitate problem-solving and decision-making*) and 5c (*Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving*).

Layer 5: Processing and Producing Deliverables

Layer 5 addresses the "metal" tools in Wing's model of computational thinking. In this layer, developers would build applications based on the algorithms created in previous layers of abstraction. Authors in an intelligent content process based on principles of computational thinking do not need to build tools. Any author who is also a programmer and decides to build an automating solution to generate deliverables following the work on previous layers of abstraction deserves special recognition. However, most writing positions do not expect that kind of work. Most likely, authors will follow the tools strategy set in layer 1 and bring all the work developed in previous layers to the tool selected. This removes the emphasis from an actual software product and highlights the work performed by humans behind the automation.

The technical communication literature shows some waves of interests for technologies included in this layer, from single-sourcing (e.g, Ament, 2003; Carter, 2003) to content management (e.g., Hart-Davidson et al., 2007; Gu & Pullman, 2009), and most recently component content management (e.g, Andersen, 2014; Batova, 2014). The current topic of interest receives attention in conference presentations and journal articles for a while, and instructors evaluate ways to teach the technology *du jour*. If an authoring process or an academic curriculum lesson starts in this processing layer, it will probably end up being too complicated for anyone who is not a programmer or developer. And if the emphasis is placed on a specific software package instead of a methodology based on human tasks of abstractions, authors in industry and academia will face the tools' acquisition problems identified by Carliner that I mentioned in the introduction to this chapter.

The main automation tool that will take our milkshake example through layer 5 is the DITA Open Toolkit – the open source implementation of the DITA standard that we discussed in previous chapters. The same results can be achieved with many DITA-aware programs (open source and commercial) with a graphical user interface, and I will also show how to process the milkshake example in Oxygen XML.

The processing layer needs the files that we have created in previous layers of abstraction, which should include the following:

- Content strategy rules developed in layer 1
- LwDITA topics authored in layer 2
- Context filters added to LwDITA topics in layer 3
- Content references added to LwDITA topics in layer 4
- XDITA map to collect topics created in layer 4.

The fictional deliverable in this example is a web version of the soda fountain operator manual that can be accessed on a device at both soda fountain locations. In this first part, the processing tool will take the map and topics and populate the *"icecream-flavor"* variable with the **<key>** value established in the XDITA map.

On a given day, the author or a manager enters the **<key>** value on the map establishing the ice cream flavor for that day. After processing, the resulting deliverable will replace all *"icecream-flavor"* placeholders found in the topics with the actual content defined in the **<key>** element on the map. Figure 8.16 in layer 4, shows an XDITA map in which the **<key>** value has been established as *"strawberry"*.

Revisiting the reference information from Chapter 5, and according to the DITA-OT documentation, the general structure for a command that builds DITA (and LwDITA) deliverables from topics and maps is shown in figure 8.17.

To generate the web version of the soda fountain operator manual, you would need to enter the command shown on Figure 8.18 on a command-line environment *inside the directory or folder where you installed the DITA-OT.*

The command has the reference to the built-in (**bin**) subdirectory and the **dita** executable program, with the *fountain.ditamap* file created in layer 4 as input, and *html5* as the delivery format. The value for the **--input** argument needs to direct the DITA-OT to the specific folder where you stored the *fountain.ditamap* file (the *path-to-file* section of the command, which must be replaced by the actual path to the file in your computer). The easiest way to record the correct file location or path is to drag and drop the file into the command line interface immediately after the **--input=** argument. That one-line command combines the map with the topics, processing the hierarchies and relationships established by the topic references and also looking for and populating any variable content placeholders. The resulting HTML files will appear automatically in the DITA-OT **out** folder, although you can specify an alternative folder in the build command with the **--output** option. Figure 8.19 shows the resulting web version of the milkshake recipe after being processed with the DITA-OT.

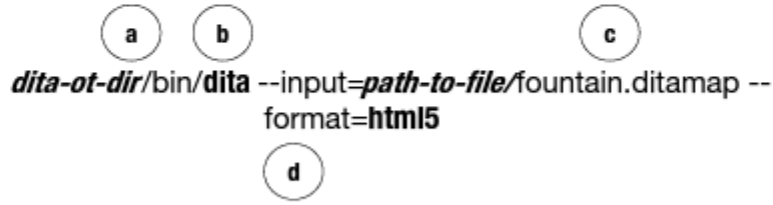
a the directory where the DITA-OT files are installed in your computer (replace *dita-ot-dir* with the full path for the DITA-OT folder).

b a subdirectory for programs that are "built-in" (*bin*) the DITA-OT. The built-in program that you need to execute is dita.

c argument for file input (e.g., a DITA/LwDITA topic or map)

d argument for format (e.g.,PDF, HTML5, etc.) and additional options.

FIGURE 8.17 General structure for a command that builds DITA (and LwDITA) deliverables with the DITA-OT.



a the directory where the DITA-OT files are installed in your computer (replace *dita-ot-dir* with the full path for the DITA-OT folder).

b a subdirectory for programs that are "built-in" (*bin*) the DITA-OT. The built-in program that you need to execute is dita.

c argument for file input (e.g., the DITA map for the soda fountain manual created in layer 4)

d argument for format (HTML5; because the soda fountain manager wants a website).

FIGURE 8.18 Command for the DITA-OT to generate a web deliverable from the soda fountain operator's manual XDITA map.

Figure 8.20 shows the resulting web version of the root beer float topic. The warning note about unplugging the blender appears in this recipe via the content reference mechanism.

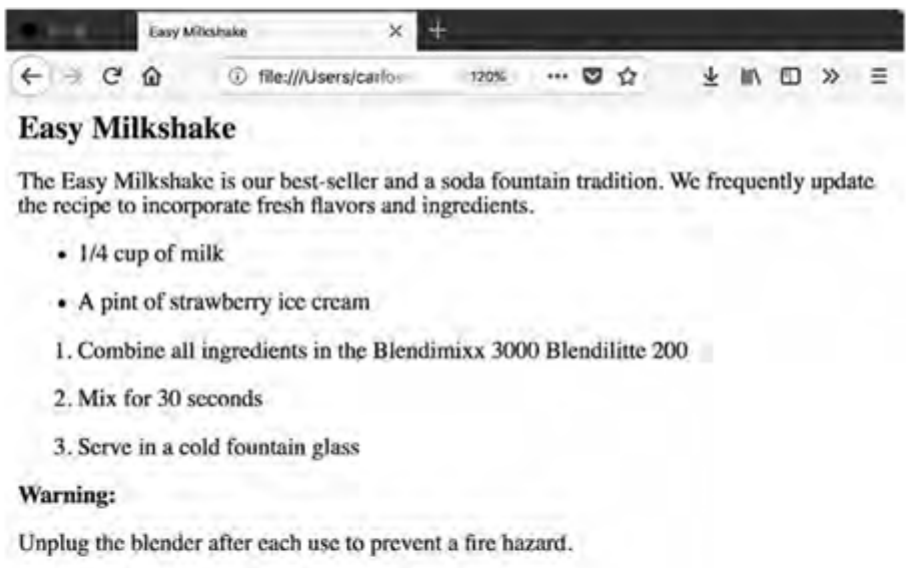


FIGURE 8.19 HTML5 transformation of the milkshake recipe processed with the XDITA map. The ice cream flavor inherited the value of "strawberry," but step 1 still shows both possible options for the blender to be used.

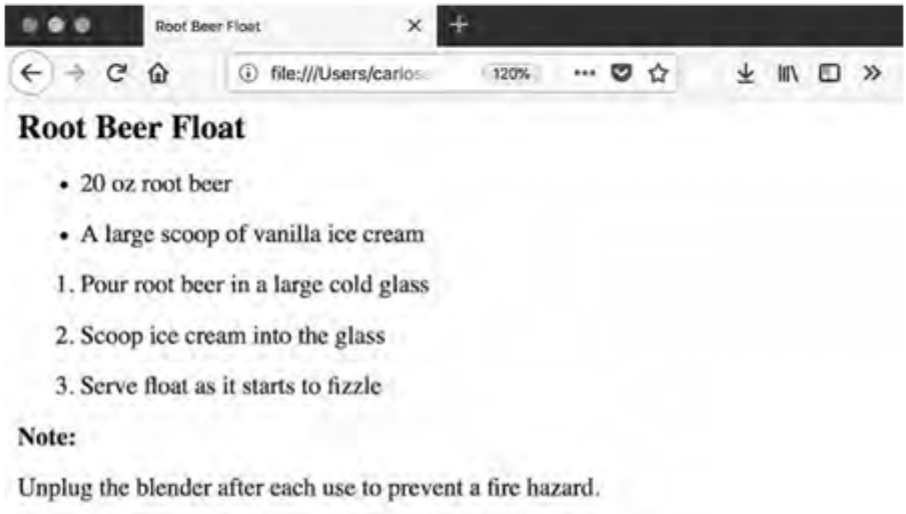


FIGURE 8.20 HTML5 transformation of the root beer float recipe processed with the XDITA map. The "warning" note appears in this topic via the conref that reuses the original element from the milkshake recipe.

A LwDITA-aware software application like Oxygen can also produce the soda fountain manual. The "Apply Transformation Scenario(s)" dialog box in Oxygen would present the user with options to produce deliverables from this LwDITA map.

If the soda fountain runs out of strawberry ice cream and needs to update the manual to reflect that the default flavor is now vanilla, the author only needs to modify one line of code inside the XDITA map, as shown in Figure 8.21.

After saving the *fountain.ditamap* with the new ice cream flavor, you can run the same DITA-OT command from Figure 8.18 without any changes. Remember to replace the *path-to-file* segment with the actual path to the file in your computer.

Figure 8.22 shows the new web version of the milkshake recipe with vanilla as the ice cream flavor.

The web versions of the milkshake recipe in Figures 18 and 21, however, still show both blender options. These deliverables are not ready for the intended users of the soda fountain manual. The next step is to apply the *if-then* filter for conditional content. This stage involves adding a new file to the collection created in these layers of abstractions. The new file will be a DITAVAL (for DITA value), which "specifies which content is included or excluded based on condition definitions" (Bellamy et al, 2012, p. 171). This DITAVAL will specify the conditional arguments for applying filters to the processing stage. If the intended deliverable is exclusively for the operators working at the food truck setting of the soda fountain, then the filter

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD LIGHTWEIGHT DITA Map//EN" "lw-map.dtd">
<map>
<topicmeta>
 <navtitle>Soda Fountain Operator Manual</navtitle>
</topicmeta>
  <keydef keys="icecream-flavor">
   <topicmeta>
    <linktext>Vanilla</linktext>
   </topicmeta>
</keydef>
  <topicref href="xdita-topics/milkshake.dita" format="xdita"/>
  <topicref href="hdita-topics/float.html" format="hdita"/>
  <topicref href="mdita-topics/bananasplit.md" format="mdita"/>
</map>
```

FIGURE 8.21 XDITA map for the soda fountain operator manual with the "icecream-flavor" variable set to vanilla.

should follow the structure of *if* **@props**="setting-foodtruck", *then* exclude **@props**="setting-downtown". This is achieved in a DITAVAL that looks like Figure 8.23.

The same DITAVAL file can work for all LwDITA authoring formats with the **@props** attribute for XDITA and the **@data-props** attribute for HDITA and MDITA extended profile. You can save this new file as *setting.ditaval* and add it to the file collection, which now includes LwDITA topics, an XDITA map, and a DITAVAL filter.

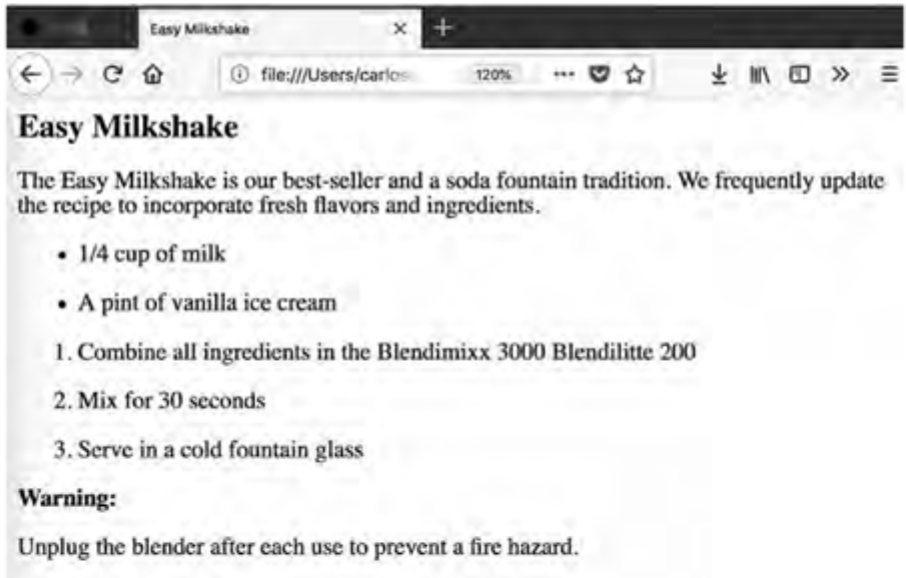


FIGURE 8.22 HTML5 transformation of the milkshake recipe with vanilla as the ice cream flavor.

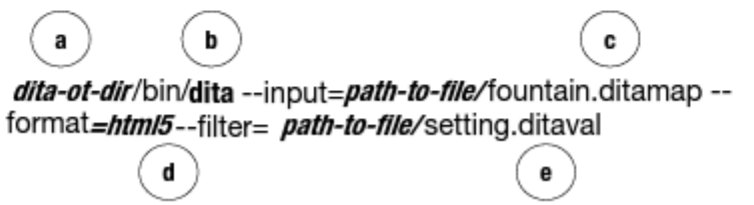
```
<?xml version="1.0" encoding="UTF-8"?>
<val>
  <prop action="exclude" att="props" val="setting-downtown"/>
  <prop action="exclude" att="data-props" val="setting-downtown"/>
</val>
```

FIGURE 8.23 DITAVAL file that excludes all references to the "downtown" setting when processing the map and topics. The two lines of code between the <val> and </val> tags reference the variables in XDITA and HDITA/MDITA extended profile syntaxes, respectively.

The marriage of the XDITA map with the DITAVAL in the DITA-OT takes place in the same one-line command but adding an option to indicate the filter location (Figure 8.24). You should also replace the *path-to-file* placeholders with the actual locations of the *fountain.ditamap* and *setting.ditaval* files in your computer.

The *filter* argument points the **dita** executable program in the direction of the DITAVAL, which will be processed with the XDITA map. The resulting web version of the milkshake recipe, which now excludes all references to the downtown location of the soda fountain, appears on Figure 8.26.

In Oxygen XML Editor, you can link a DITAVAL to a LwDITA map when configuring the transformation scenario by clicking on the *Filters* tab. The Oxygen user manual includes information on applying filters to DITA transformations[2].



a the directory where the DITA-OT files are installed in your computer (replace *dita-ot-dir* with the full path for the DITA-OT folder).

b a subdirectory for programs that are "built-in" (*bin*) the DITA-OT. The built-in program that you need to execute is dita.

c argument for file input (e.g., the DITA map for the soda fountain manual created in layer 4)

d argument for format (HTML5; because the soda fountain manager wants a website).

e argument for filter (the DITAVAL file that excludes all references to the "downtown" location).

FIGURE 8.24 Command for the DITA-OT to generate a web deliverable from the soda fountain operator's manual XDITA map that excludes references to the "downtown" location.

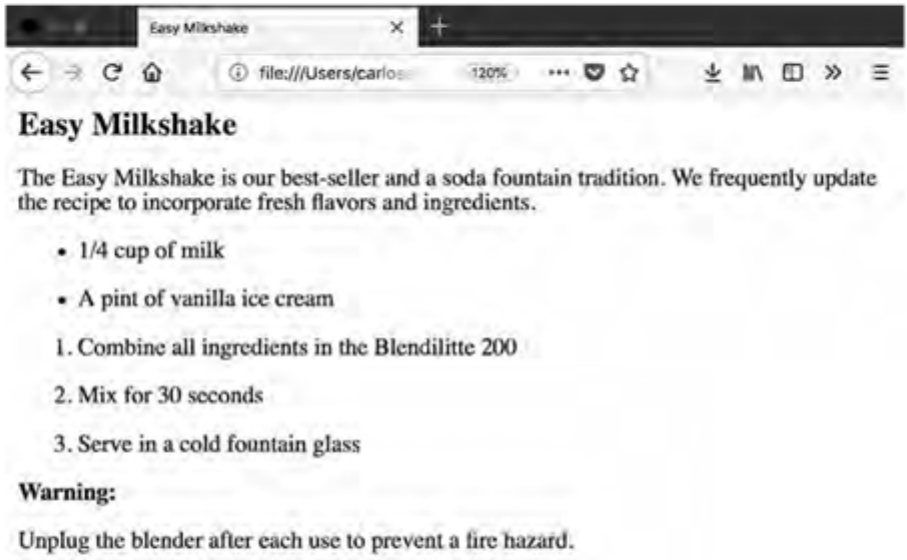


FIGURE 8.25 Filtered web deliverable of the milkshake recipe that excludes all references to the "downtown" setting.

If the setting needs to change, you only need to update the DITAVAL and process again with the same DITA-OT one-line command or applying the transformation scenario in Oxygen. For larger content repositories, this build process can be automated with scripts and algorithms, which are beyond the scope of *Creating Intelligent Content with Lightweight DITA* (aimed primarily at authors and instructors of introductory LwDITA workflows).

Building deliverables with the DITA-OT or LwDITA-aware tools is only one stage of this layer. The soda fountain will need a technology-based solution to store source code, enable authoring and processing, and deliver information products to end users (on web or print platforms). These layers of abstraction reflect the academic privilege behind their creation, as in a classroom environment my students do not have to deal with workplace clients and users with real needs. Even in client projects for internships or service learning purposes, deadlines and deliverables are flexible when they involve undergraduate students as content creators. Thus, this layer does not get involved with advanced software for component content management and presentation of deliverables. In my courses at Virginia Tech, students learn to host their source code and track versions of it with GitHub repositories, and their processing is automated with the DITA-OT or Oxygen XML. GitHub Pages is the default hosting service for web deliverables produced in this layer. Graduates of the Professional and Technical Writing program who work in industry, non-profits, or government agencies authoring for intelligent content workflows learn their worksite-specific tools

as part of job training programs, and the layers of abstraction presented in this chapter make that learning process easier, particularly for graduates from a department of English. Teaching intelligent content (or even DITA) using a commercial CCMS, following its professionally-developed video tutorials and typing in a graphical user interface, is a professional skill that might benefit a handful of students and create limitations for many others who end up in environments that do not use that specific tool.

The tasks involved in this layer are related to stages of the content-development lifecycles featured earlier in the previous chapter. Layer 5 tasks focus on Bailie & Urbina's *publish* stage. They also cover some of the work included in Andersen's *Developing the technology strategy* and *Creating structured content*. Contrasting layer 5 with the ISTE standards for assessing computational thinking in students, some of its tasks could comply with sections of standard 5d (*Students understand how automation works and use algorithmic thinking to develop a sequence of steps to create and test automated solutions*) with limitations.

Layer 6: Preparing for the Future

Now that the soda fountain has an operator manual that adapts to different locations, reuses warning content among topics regardless of authoring format, and incorporates the ice cream flavor of the day to all its recipes with a one-line command, it's time to think about the future. LwDITA allows scalability as represented by additional publication channels, new audience and context requirements, and authors from different disciplines and writing traditions joining the process. Additional publication channels can include those already supported by the DITA-OT or other existing LwDITA-aware software applications (EPUB, mobile web devices, PDF, Microsoft Word, and others) or innovative approaches to user documentation like the conversational patterns for chatbots or agents like an Amazon Echo. New audience and context requirements can come from expert or novice fountain operators who need different levels of detail on their recipes, or additional settings for the soda fountain with different equipment and ingredients availability. Authors from different writing traditions can include those already represented in the initial LwDITA authoring formats (XML, HTML5, and Markdown) and those in formats considered for future releases of the LwDITA standard (e.g., JSON and Microsoft Word). This layer also includes a return to layer 1 for an iterative process of improvement and revision.

In general, this layer focuses on making the content and structures developed in previous layers as future-proof as possible. This is accomplished by the emphasis on open standards instead of proprietary software solutions. DITA, LwDITA, HTML5, and even CommonMark/GitHub Flavored Markdown are

maintained by international standards' consortiums. Their continuous development guarantees applicability beyond the lifespan of a specific tool or app release. Furthermore, it also enables interoperability with new standards, like the International Standard for Intelligent Information Request and Delivery (iiRDS)[3], which has been heralded as the standard of Industry 4.0 and Internet of Things workflows.

The tasks involved in this layer are related to several stages of the content-development lifecycles featured earlier in the previous chapter. Layer 6 tasks focus on the characteristics of extensible and iterative from Bailie & Urbina's content model, which includes "a loop back to analysis for the next cycle" (2013, p. 236). The tasks in layer 6 also cover some of the work included in Andersen's *Managing change* stage. Contrasting layer 1 with the ISTE standards for assessing computational thinking in students, some of its tasks could comply with sections of standards 5a (*Students formulate problem definitions suited for technology-assisted methods such as data analysis, abstract models and algorithmic thinking in exploring and finding solutions*), 5c (*Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving*), and even 5d (*Students understand how automation works and use algorithmic thinking to develop a sequence of steps to create and test automated solutions*) .

Notes

1 "1/4 cup of milk," as the XDITA version of the recipe presented in Figure 1 shows, is not really the same as "¼ (one quarter) cup of milk." Representing fractions in DITA can be properly achieved with snippets of the W3C standard MathML (https://www.w3.org/Math/). In order to keep this example as simple as possible, I avoided the complexity of adding MathML syntax to the XDITA topic.

2 https://www.oxygenxml.com/doc/versions/20.0/ug-author/topics/dita-map-edit-filters.html

3 https://www.parson-europe.com/en/knowledge-base/427-iirds-new-delivery-standard-technical-documentation.html

References

2.2.4.2.1 Conditional processing attributes (2018, June 19). Retrieved from http://docs.oasis-open.org/dita/dita/v1.3/errata02/os/complete/part3-all-inclusive/archSpec/base/conditional-processing-attributes.html#conditional-processing-attributes

Albers, M. J. (2003). Single sourcing and the technical communication career path. *Technical Communication*, 50(3), 335–344.

Ament, K. (2003). *Single sourcing: Building modular documentation*. Norwich, NY: William Andrew.

Andersen, R. (2014). Rhetorical work in the age of content management: Implications for the field of technical communication. *Journal of Business and Technical Communication*, 28(2), 115–157.

Bacha, J. (2009). Single sourcing and the return to positivism: The threat of plain-style, arhetorical technical communication practices. In G. Pullman & B. Gu (Eds.) *Content management: Bridging the gap between theory and practice* (pp. 143–159). Amityville, NY: Baywood.

Bailie, R.A. (2014). Content strategy. In S. Abel & R.A. Bailie (Eds.), *The language of content strategy* (pp. 14–15). Laguna Hills, CA: XML Press.

Bailie, R.A. & Urbina, N. (2013). *Content strategy: Connecting the dots between business, brand, and benefits*. Laguna Hills, CA: XML Press.

Baker, M. (2013). *Every page is page one: Topic-based writing for technical communication and the web*. Laguna Hills, CA: XML Press.

Batova, T. (2014). Component content management and quality of information products for global audiences: An integrative literature review. *IEEE Transactions on Professional Communication*. 57(4), 325–339.

Bellamy, L., Carey, M., & Schlotfeldt, J. (2012). DITA best practices: A roadmap for writing, editing, and architecting in DITA. Upper Saddle River, NJ: IBM Press.

Carliner, S. (2010). Computers and technical communication in the 21st century. In R. Spilka (Ed.), *Digital literacy for technical communication: 21st century theory and practice* (pp. 21–50). New York, NY: Routledge.

Carter, L. (2003). The implications of single sourcing on writers and writing. *Technical Communication*, 50(3), 1–4.

Clark, D. (2007). Content management and the separation of presentation and content. *Technical Communication Quarterly*, 17(1), 35–60.

Clark, D. (2016). Content strategy: An integrative literature review. *IEEE Transactions on Professional Communication*, 59(1), 7–23.

Creekmore, L. (2014). Metadata. In S. Abel & R.A. Bailie (Eds.), *The language of content strategy* (pp. 28–29). Laguna Hills, CA: XML Press.

Flower, L. (1989). Rhetorical problem solving: Cognition and professional writing. In M. Kogen (Ed.), *Writing in the business professions* (pp.3–36). Urbana, IL: NCTE.

Gallagher, J.R. (2017). Writing for algorithmic audiences. *Computers and composition*. 45, 25–35.

Getto, G. & St. Amant, K. (2014). Designing globally, working locally: Using personas to develop online communication products for international users. *Communication Design Quarterly*. 3(1), 24–46.

GitHub Flavored Markdown Spec. (2017, August 1). Retrieved from https://github.github.com/gfm/

Hackos, J. T. (2011). *Introduction to DITA: A user guide to the Darwin Information Typing Architecture including DITA 1.2* (2nd edition). Denver, CO: Comtech Services, Inc.

Halvorson, K. & Rach, M. (2012). *Content strategy for the web* (2nd Ed.). Berkeley, CA: New Riders.

Harrison, N. (2016). Content variables. In R. Gallon (Ed.), *The language of technical communication* (pp. 66–67). Laguna Hills, CA: XML Press.

Hart-Davidson, W. et al. (2007). Coming to content management: Inventing infrastructure for organizational knowledge work. *Technical Communication Quarterly*. 17(1), 10–34.

ISTE. (2018). ISTE standards for students. Retrieved from http://www.iste.org/standards/for-students

Pringle, A., & O'Keefe, S. (2009). *Technical writing 101: A real-world guide to planning and writing technical content*. Durham, NC: Scriptorium Press.

Rothrock, L. & Narayanan, S. (2011). *Human-in-the-loop simulations: Methods and practice*. London: Springer-Verlag.

Swarts, J. (2010). Recycled writing: Assembling actor networks from reusable content. *Journal of Business and Technical Communication*. 24(2), 127–163.

Swisher, V. (2014). *Global content strategy: A primer*. Laguna Hills, CA: XML Press.

Vazquez, J. (2016). Conditional content. In R. Gallon (Ed.), *The language of technical communication* (pp. 64–65). Laguna Hills, CA: XML Press.

Wachter-Boettcher, S. (2012). *Content everywhere: Strategy and structure for future-ready content*. Brooklyn, N.Y: Rosenfeld Media.

Willerton, R. (2015). *Plain language and ethical action: A dialogic approach to technical content in the 21st century*. New York, NY: Routledge.