1

REVISITING THE FUTURE OF TECHNICAL COMMUNICATION

James Mathewson knows a thing or two about technical communication. In his role as Distinguished Technical Marketer at IBM, James, a graduate of the Master of Science in Scientific and Technical Communication from the University of Minnesota-Twin Cities, was looking for interns who could work in a structured authoring environment. James took to Twitter to express his frustration after coming back empty-handed from his search for interns.

> Yes. Writing reusable content is a rare skill. Would that they taught it in college. Essays from whole cloth is more the thing.
>
> (Mathewson, 2016)

When he talks about "reusable content," Mathewson is referring to the "practice of using content components in multiple information products" (Eberlein, 2016 p. 54). In many technical communication practices, "efficient content reuse does not involve copy-and-pasting; instead it uses *transclusion*, whereby content is authored in one location and used by reference in other locations" (Eberlein, 2016 p. 55). The "essays from whole cloth" reference is a nod to the book-oriented kind of writing using a word processor that most academic programs in technical communication and content development teach as standard practice.

It was not supposed to be like this.

In 2012, the Adobe Technical Communication Suite (TCS) team distributed on several social media channels a video titled "The Future of Technical Communication." The video used stop-motion animation and fast-draw techniques to summarize the features of "Adobe's Tools and Services" for technical communication. As a pair of rapidly animated hands assembles Lego pieces (Figure 1.1), the video's narrator describes that "for some, (the future of technical

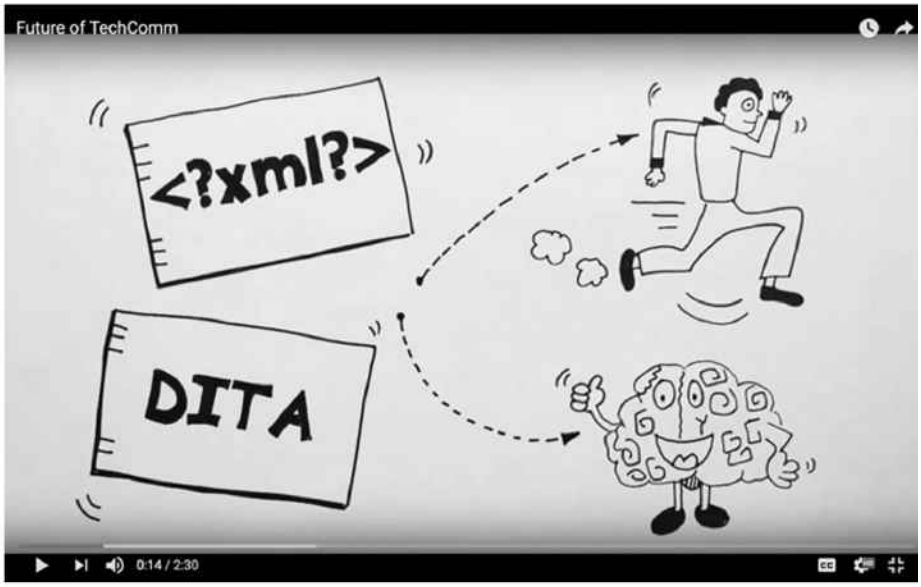


FIGURE 1.1 Screen capture from the video "The Future of Technical Communication," by Adobe Technical Communication Suite. This specific frame displays the future of our profession as "more and mor structured content and the ability to work faster and smarter with XML and DITA constructs."

communication) is all about more and more structured content and the abilit to work faster and smarter with XML and DITA constructs." Approaching th end of its 2:30-minutes runtime, the video claims that "it is most certainly a exciting future to be in" (AdobeTCS, 2012).

The "future" of technical communication described in the Adobe vide focuses on what Rebekka Andersen describes as "structured content that highly adaptable and portable and can be configured on the fly in respons to specific user requests" (2014, p. 116). Andersen adds that this type of cor tent supported by topic-based information design "has been given variou names, including intelligent content, nimble content, smart content, portabl content, and future-ready content." Rockley, Cooper, and Abel describe inte ligent content as "designed to be modular, structured, reusable, format fre and semantically rich and, as a consequence, discoverable, reconfigurabl and adaptable" (Rockley et al., 2015, p. 1). In academia, scholars have praise Extensible Markup Language (XML) as the foundation for information reus single sourcing, and, particularly, content management for more than a dec ade. Publications from the 2000s heralded that "technical communicators wi probably face a day when all organizational documents are saved in XML for mat" (Sapienza, 2002, p. 156) and argued that "technical communicators shoul be able to write, edit, and manage XML documentation, including XML tag

document type definitions (DTDs), XML schemas, and Darwin Information Typing Architecture (DITA)" (Gesteland McShane, 2009, p. 74).

The future of technical communication, as seen from industry and academia, was supposed to cover the type of structured writing for reuse that James Mathewson was looking for.

It is not as though the academic side of technical communication is stagnant. On the contrary, scholars in the field have expanded the work of technical communication into domains and scenarios like healthcare, policy-making, and social justice, among others, that were closed to writing researchers in the recent past. James's tweet started a discussion about the many topics covered in courses and programs in technical communication. Researchers work with practitioners that include community partners, government officials, and medics. However, "many of the pedagogical concerns of academic instructors in professional and technical communication have changed little over the past decade, even as practitioner discourse has continued to spin off in the direction of content strategy and similar areas" (Clark, 2016, p. 19). Academic discussion related to the implementation and innovation of intelligent content has been slow, as evidenced by the titles and abstracts of presentations and publications from 2012 to 2018 in relevant conferences and journals, respectively.

Traditionally, and as mentioned in the Adobe video, intelligent content workflows for technical communication rely on structure provided by Extensible Markup Language (XML) and some of its specific grammars like DocBook, S1000D, and DITA (Cowan, 2010; Glushko & McGrath, 2008; Hackos, 2011, among others). Although the benefits of a well-planned and implemented intelligent content workflow built on XML or DITA are undeniable, this type of solution presents significant challenges that have repercussions in the academic and workplace facets of the profession.

Technical communication consultants and practitioners frequently report on success stories and challenge narratives from implementing and managing XML-based content projects in conferences like DITA/Content Management Strategies and LavaCon. Furthermore, blogs and social media postings from content professionals create and maintain an active online community of discussion and recommendations. Because of proprietary practices, it would be impractical to poll private companies, government agencies, and non-profit institutions worldwide to tally a number of DITA and XML users. Nevertheless, some attempts at quantifying and documenting usage figures do exist. For example, the website "DITA Writer" maintains a list of companies using DITA based on reports from social media channels. As of summer 2018, the list included 724 companies, excluding consulting and training firms (DITAWriter, n.d.).

Adoption numbers and success stories should not hide that the evolution of intelligent content takes place on a slightly rocky path; even in practitioner circles, there is pushback and criticism against XML and its relationship with technical communication. In blogs and social media exchanges, some practitioners have

questioned the status of XML, and DITA, as the main markup language for infor mation products. While acknowledging DITA's effectiveness as a replacement fc large user manuals in complex industries, a few authors lament that "this form c structured content can feel cold and clinical, especially to those from the editoria or marketing side of content" (Wachter-Boettcher, 2012, p. 20). Others argue tha in the world of computing code verbose languages are becoming obsolete, bu intelligent content still relies on XML and its nested tag structures:

> What are we seeing? Simplification. Ease of use. A learning curve that get less steep every time. Languages that drop features that aren't used, c aren't used often. And what has techcomm poured resources into? DITA An arcane, overly complex language with a massive learning curve tha requires specialized tools.
>
> (Kaplan, 2014

Those are valid concerns. DITA, as it evolves as an open standard, needs t address them and learn from its users. This chapter presents an overview of th evolution of DITA – an XML-based "end-to-end architecture for creating an delivering modular technical information" (Priestley et al., 2001, p. 354). Th future of technical communication still involves XML; therefore, the follow ing sections include brief introductions to Extensible Markup Language an the Darwin Information Typing Architecture, providing examples of the mai content and collection types included in the latter. Then, the chapter explain the need for a simplified version of DITA that allows authors to contribute t intelligent content ecosystems writing in markup languages that do not requir complex XML structures. Lastly, the chapter focuses on strategies for adoptin a DITA workflow in three simplified authoring formats, which are connected t principles of computational thinking and emphasize the importance of huma abstraction before machine automation.

An Introduction to XML in Technical Communication

Although an author does not even need to be familiar with XML in order t create information products in the workflows introduced in this book, th ideas presented in *Creating Intelligent Content with Lightweight DITA* ar rooted in DITA and XML. As a result, I cannot leave out working definition and background for context when introducing the standard and its evolutio as a problem-solving methodology for technical communication. This sectio will be especially useful to readers who have not ventured into standards an markup languages for intelligent content.

First, I define XML for the specific context of *Creating Intelligent Conter with Lightweight DITA*. Definitions and descriptions of the Extensible Marku Language abound in academic and practitioner publications. A search for "XMI

on amazon.com reveals more than 1,100 results under the "Books: Computers & Technology: Programming Languages" category. In the realm of rhetoric studies, Applen and McDaniel wrote an important text about XML and its implications for rhetorical work. In one of the contrasting moves of their definition, they posit XML as different from Hypertext Markup Language (HTML) and other typesetting and formatting syntaxes because it enables the processes of "identifying, separating, and recombining" content for different purposes (Applen & McDaniel, 2009, p. 42). Content in XML, for example, can be tagged as a product name, a procedure, a works cited entry, or any other part of a technical or professional document. Once those parts or components are tagged, they can be "assembled" in different ways for use in a variety of deliverables and media. New documents need not be written for each purpose, and one single change can be reflected accurately across all the information materials produced by a given company.

Let me introduce you here to Pedro, a chef and restaurateur who is going to be the main user-author in this book's examples. Chef Pedro has a background in marketing and culinary arts. He is comfortable with technology and social media, but he has never worked with any type of computer code. He is looking for a way to standardize the recipes used by his staff in the several kitchens he manages. As a franchise owner, he wants to ensure that the food in his restaurants is consistent, and his kitchen staff needs up-to-date documentation with recipes and techniques adopted in all of his restaurants. At the same time, the menus in his restaurants need to reflect some of that content, which is also featured on a website, mobile app, and an electronic book he sells describing the history and process of his restaurants. Chef Pedro, like many content owners in this world, has been using Microsoft Word for most of his information products, and the publications department in his company takes some of those Word files and, through long sessions of copy and pasting, produces menus and flyers in Adobe InDesign and maintains the restaurant's website using WordPress. When a change needs to be reflected in those information products (e.g., the kitchen introduces a new special or the office's telephone number should be updated), Chef Pedro makes the changes in his original Word files, and then the publications team has to manually update the menus, flyers, and website that borrow content from the Word master source.

Someone mentioned structured authoring with XML and that caught Chef Pedro's attention. The promise: a centralized content repository structured in XML can be the source for many user deliverables. And the production and update of those deliverables can be automated. No more copy and paste! Now, if Pedro were to open a text editor in his computer and start typing XML tags to structure his content, he probably would not know where to go after typing for a few minutes. To create information deliverables ready for human consumption, XML files need to go through a process of publishing that, according to O'Keefe, can involve the following roles, which "in a small group, one person may hold any or all" (2009, p. 14):

- Document architect, who defines and implements document structure
- Template designer, who establishes the look and feel of content deliverable
- Writer, who creates content
- Technical editors, who can focus on word choice, grammar, and overa organization
- Production editors, who will get involved in defining the transformatio files that assign formatting based on structure.

For someone like Chef Pedro (and even for many of my technical writin students), creating the tags to structure content in XML can be an easy-to learn task. On the other hand, producing information deliverables based o that structure (say, a webpage with a specific recipe) is not necessarily simpl when implementing a custom XML content type. That's when a standard lik DITA can help. Pedro could buy a software package to structure his recipes an create an online cookbook; however, in a previous stage of his career he wa involved in a marketing project that depended on a commercial applicatior When the developer stopped updating the application and it became obsolet Pedro's content was locked and required an expensive and time-consumin process of conversion to be rescued in another program. A chef might not b familiar with scholarship in writing studies, but authors like Karl Stolley, i his influential "The Lo-Fi Manifesto," have argued for the adoption of ope standards over software packages, identifying it as the only way to ensur that "digital works should long outlast the software that played a role in thei creation" (Stolley, 2008).

This kind of data tagging and manipulation that XML allows is at the hea of intelligent content. From the practitioner side of technical communicatior O'Keefe points out that XML "defines a standard for storing structured conter in text files" (2009, p. 15). O'Keefe's extended definition includes the followin features of XML:

- It is a markup language, which means that content is enclosed by tags.
- Its element tags are enclosed in angle brackets (<element>This is elemer text</element>).
- It does not provide a set of predefined tags. Instead, authors define thei own tags and their relationships. (O'Keefe, 2009, p. 15)

Additionally, O'Keefe provides a sample XML file that presents a recipe fc making marinara sauce. (O'Keefe, 2009, p. 16) (Figure 1.2).

This process of tagging, commenting, and nesting data will look familiar t readers who have used HTML or another markup language. O'Keefe's recipe[1] fea tures a main container element (**<Recipe>**), which includes several sub-element (**<Name>**, **<IngredientList>**, and **<Instructions>**). Some of these sub-element

```
<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>
```

FIGURE 1.2 Sarah O'Keefe's sample recipe for marinara sauce in XML. The structured recipe keeps content organized and should help computers understand and manipulate elements and attributes. Human readers, however, still need to wait for a publishing process that will produce content deliverables that do not look like computer code.

hold actual content (e.g., **<Name>**), and others nest additional levels of elements and, eventually, content (e.g. the "parsley" **<Item>** inside **<Ingredient>** inside **<IngredientList>**). XML also uses attributes to present additional information about data and content (see the **@Cuisine** and **@Author** attributes in the main **<Recipe>** element). If you're unfamiliar with XML structures, don't worry; the relevant XML elements for this book will be discussed in detail in subsequent chapters. For now, just focus on the fact that XML uses tags to identify the types of rhetorical "moves" that the content on the page represents.

Just like Chef Pedro, who wants to create a structured template for all entries in his recipe guide, information developers at IBM started looking at XML in the late 1990s to organize technical content. One of the solutions that came out of those explorations at IBM was the Darwin Information Typing Architecture.

Much Ado About DITA

In an interview for the website "DITAWriter," Don Day, one of the origina developers of the DITA standard, chronicled the origins of his XML exper ments while working for IBM in the last decade of the 20th century as follows:

> With the advent of XML as a new markup standard in 1998, the Custome and Service Information (C&SI) group began adopting a Tools an Technology mantra under Dave Schell who was the strategy lead. B 1999, Dave was aware of my participation as IBM's primary representa tive with the XSLT and CSS standards activities at the World Wide We Consortium, and I delivered a presentation at a formative meeting i California that forecast the possibility of XML to solve IBM's still-linge ing problems with variant tools and markup usage.
>
> (Day, quoted in DitaWriter, 201

DITA consists of a set of design principles for creating "information-typed modules at a topic level and for using that content in delivery modes such a online help and product support portals on the Web. (Day et al., 2001). Da explained that, when naming the standard, DITA "represented a great deal c messaging in a compact and memorable acronym:"

- **Darwin**: for specialization and how things could "evolve" from a base
- **Information Typing**: for representation of knowledge as typed units
- **Architecture**: a statement that this was not just a monolithic design but an exter sible tool that could support many uses (Day, quoted in DITAWriter, 2016).

IBM eventually donated DITA as an open standard, which is currently mair tained by the non-profit consortium OASIS. DITA, however, "has evolve substantially since that initial donation to encompass a very wide scope of require ments indeed" (Kimber, 2012, p. 6). At the OASIS DITA Technical Committe the standard continually evolves with the purpose "to define and maintain th Darwin Information Typing Architecture (DITA) and to promote the use of th architecture for creating standard information types and domain-specific marku vocabularies" (OASIS Darwin Information Typing Architecture TC, n.d.). Hacko summarizes the key benefits of DITA for technical communicators as follows:

- A fully tested DTD or schema for XML-based authoring
- A community of developers investing in improvements to the DITA mod
- An open source toolkit you can use to produce your own output in multipl media without having to invest in proprietary tools
- A thoroughly developed approach to information development origina ing with OASIS and now encompassing many other companies, large an small, that find value in a standards-based approach (2011, p. 9).

Additionally, from a perspective addressing the needs of managers and supervisors, Hackos presents a list of DITA's business advantages, which suggest that the standard will (promote) the reuse of information quickly and easily across multiple deliverables, reduce the cost of maintaining and updating information, enable continuous publishing, share information across the global enterprise, reduce the cost of localization, and reduce the technical debt caused by inadequate, incorrect, and unusable legacy information (Hackos, 2011, p. 10).

For Chef Pedro, adopting a standard like DITA simplifies the process of structuring content and producing information deliverables for human users. Instead of designing custom tags and depending on a publishing team to design templates and validation tools, he can use DITA's content types and take advantage of the benefits listed by Hackos. For an author, the main benefit from Hackos's list is in the "fully tested DTD or schema for XML-based authoring," which I discuss in the next section.

DITA Content Types: More than Templates

DITA's "fully tested DTD or schema for XML-based authoring" comes from the *Information Typing* part of its name. Content in DITA is presented as units or individual XML files that conform to pre-established types or models. Those pre-established content types are enforced by files that are commonly referred to as DTDs, although DTD is only one of the markup languages used to verify the structure of those files. XML, and as a consequence DITA, files can also be validated by XML Schema and RELAX NG (Regular Language for XML Next Generation) files.

If that validation process sounds a little too complicated, as an author all you need to know is that the content types enforced by the DITA standard create information topics. A topic is "a self-contained unit of information. An effective topic covers only one subject. Each topic is long enough to make sense on its own, but short enough to stick to one point without expanding into other subjects" (Bellamy et al. 2012, p. 8), and can be defined as "small independent piece of information on a single subject" (Baker, 2013, p. 71). Topic-based writing is described as "authoring an information set as a collection of discrete units called *topics*, rather than as a whole book or help system" (Baker, 2016 p. 52). This kind of writing is the basis of several technical communication and intelligent content practices and techniques, including component content management systems (CCMS), which are "a centralized system that helps organizations capture, manage, store, preserve, and deliver topic-based content (components)" (Kerzreho, 2016 p. 60), and single sourcing, which can be defined as the practice of "creating content once, planning for its reuse in multiple places, contexts, and output channels" (White, 2016 p. 56).

While authors certainly can work on a topic-based environment without DITA (see Baker, 2013), the term *topic* is frequently associated with this open standard, as explained in the following quote from Andersen & Batova:

> Content components are the building blocks of information product While terms such as granules, modules, and units are commonly use to describe these blocks, the term topic has gained the most traction i the past few years, particularly in the trade literature. Topic derives fro the widely adopted open content standard known as Darwin Informatio Typing Architecture (DITA), which defines a common structure for cor tent that promotes the consistent creation, sharing, and reuse of conten
> (Andersen & Batova, 2015, p. 25

The literature focuses on three topic types that "represent the vast majority c content produced to support users of technical information" (Hackos, 2011, p.7 concept, task, and reference, which Pringle & O'Keefe define succinctly as follow

- Concept: contains background information and examples
- Task: includes procedures ("how to" information)
- Reference: describes commands, parameters, and other features (Pringle & O'Keefe, 2009, p. 235).

For authors of technical content, these foundational topic types provid constraints and structures beyond a presentation-oriented template. In DITA authors can create consistent topics to assemble collections of information wit elements that can be reused even at the phrase level. For example, a concep could be an introduction to the sauces section in Chef Pedro's recipe book, whil tasks can provide recipes for specific salsas and condiments (he can write a ste about dicing tomatoes once and reuse in all the recipes that need it!), and a re erence topic can list common techniques and tools for preparing ingredients.

In practical terms, DITA's topic types include XML tags for content "moves or strategies (such as a short description, steps, and examples) frequently use in technical publications. Pure XML (as we saw in the marinara sauce example does not provide a defined set of tags, but DITA does offer a catalog of element and attributes relevant for technical communicators. Although in Chapter 5 and 6 I will further analyze the DITA tags and attributes, for the scope c this introductory section, I structured and tagged O'Keefe's recipe for mari nara sauce as a DITA task (Figure 1.3). You can download these code sample from the *Creating Intelligent Content with Lightweight DITA* GitHub repositor (https://github.com/carlosevia/lwdita-book).

The recipe has a strong structure with visible sections. The **<task>** element oper the topic announcing "this is a task," and it contains elements like the following:

- **<shortdesc>** with a summary of the topic's contents
- **<prolog>** with some information about the recipe's author and category
- **<prereq>** with a list of the ingredients needed for the recipe
- **<steps>** with a collection of individual **<step>** elements containing a comman (**<cmd>**) with an action verb.

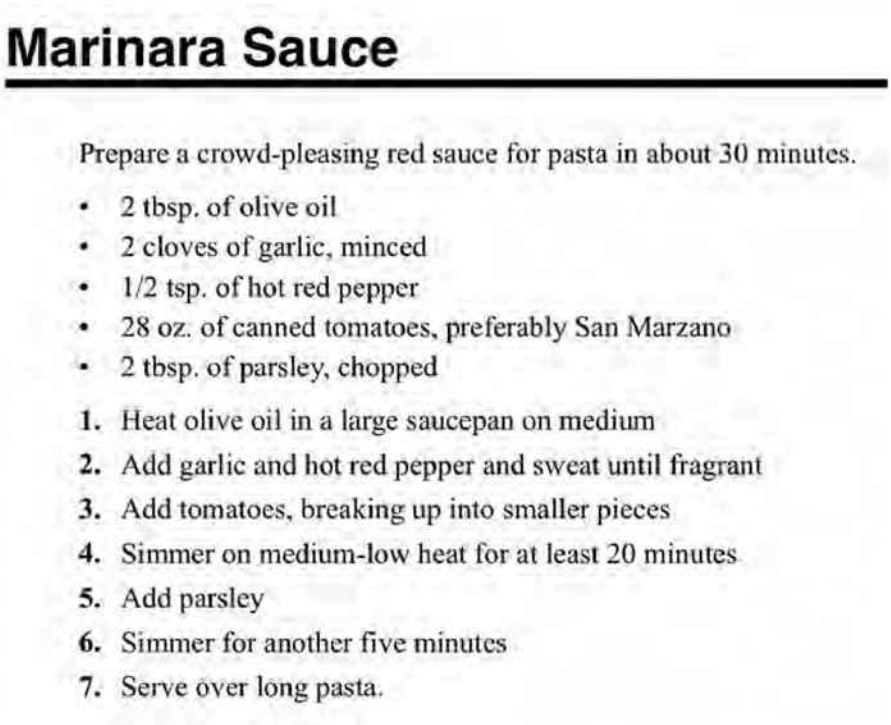
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE task PUBLIC "-//OASIS//DTD DITA Task//EN" "task.dtd">
<task id="t-marinara">
    <title>Marinara Sauce</title>
    <shortdesc>Prepare a crowd-
pleasing red sauce for pasta in about 30 minutes.</shortdesc>
    <prolog>
        <author>Unknown</author>
        <metadata>
            <category>Italian</category>
        </metadata>
    </prolog>
    <taskbody>
        <prereq>
            <ul>
                <li>2 tbsp. of olive oil</li>
                <li>2 cloves of garlic, minced</li>
                <li>1/2 tsp. of hot red pepper</li>
                <li>28 oz. of canned tomatoes, preferably San Marzano</li>
                <li>2 tbsp. of parsley, chopped</li>
            </ul>
        </prereq>
        <steps>
            <step>
                <cmd>Heat olive oil in a large saucepan on medium</cmd>
            </step>
            <step>
                <cmd>Add garlic and hot red pepper and sweat until fragrant
</cmd>
            </step>
            <step>
                <cmd>Add tomatoes, breaking up into smaller pieces</cmd>
            </step>
            <step>
                <cmd>Simmer on medium-low heat for at least 20 minutes</cmd>
            </step>
            <step>
                <cmd>Add parsley</cmd>
            </step>
            <step>
                <cmd>Simmer for another five minutes</cmd>
            </step>
            <step>
                <cmd>Serve over long pasta.</cmd>
            </step>
        </steps>
    </taskbody>
    </task>
```

FIGURE 1.3 Marinara sauce recipe as a DITA task. The standard's "fully tested DTD or schema for XML-based authoring" includes commonly used elements or moves in technical publication. Thus, the recipe collector does not have to invent custom tags.

Hackos mentioned "an open source toolkit you can use to produce your own output in multiple media without having to invest in proprietary tools" (2011, p. 9). Using that toolkit, known as the DITA Open Toolkit (OT), or a software tool that uses the Open Toolkit, Chef Pedro can produce quick information deliverables. If a sous-chef needs a printable PDF version of the recipe, Pedro can produce it without hiring a template designer (Figure 1.4). And if a different member of the

kitchen team requires a web version of the recipe, the DITA-OT also provide that option and allows the author to link to a Cascading Style Sheet (CSS) fil for formatting and design (Figure 1.5). In this introductory chapter I will onl present the results of transformations using the DITA-OT, but in Chapter 5 will focus on specific steps for conducting those transformations.

Concept, task, and reference are, for many authors and their manager essential to DITA. Yet, the big-picture ideas of topic-based information an component content management go beyond the actual topic types enforced b the DITA standard and DITA-aware tools. Although concept, task, and ref erence are still defined types in the DITA standard, the official specificatio for DITA 1.3 also includes topic types for troubleshooting, which "provide markup for corrective action information such as troubleshooting and aları clearing" (2.7.1.6 Troubleshooting topic, 2016) and glossary, which "defines single sense of one term" (2.7.1.7 Glossary entry topic, 2016). Actually, the al inclusive edition of the DITA 1.3 standard has 26 document types (predefine document, or even genre, templates) and 621 element types (placeholders c structures for specific content moves). Even in its base edition, DITA 1.3 has document types and 189 element types. And authors should not underestimat



Marinara Sauce

Prepare a crowd-pleasing red sauce for pasta in about 30 minutes.

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

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

FIGURE 1.4 Marinara sauce DITA task transformed to a PDF deliverable. This simple output did not require a dedicated stylesheet or choices about pagination and typography.

FIGURE 1.5 Marinara sauce DITA task transformed to an HTML5 deliverable. This sample output includes a link to an external CSS file for formatting.

the importance of a generic topic, defined as "the basic unit of authoring and reuse" (2.2.1.1 The topic as the basic unit of information, 2016), which is the only information type included in the base edition of the DITA standard. The generic topic, with its simple tags for paragraphs and lists, can mark up a marketing blog post. A basic topic can also include a flexible procedure, or even provide pointers for a web-based product tour with some JavaScript processing, among many other applications. Additionally, the versatile task topic in DITA 1.3 can take the shape of a general task topic, a strict task, or a machinery task topic, depending on the context and purpose of usage. DITA 1.3 also includes topic types designed for learning and training projects: learning plan, learning overview, learning content, learning summary, and learning assessment.

If those pre-established topic types were not enough for a particular writer and context, DITA topics can be *specialized* to create information types unique to any intelligent content domain. At the topic level, and without getting too deep into the process of DITA specialization, just as **<concept>** and **<task>** are specialized from the original generic **<topic>**, **<task>** could specialize into **<recipe>**. For example, Chef Pedro is going to need to provide ingredients for his recipes, so he could create an ingredients specialization to include this specific element (i.e., **<ingredients>** as a specialization of **<prereq>** for pre-requisites). This exercise in markup flexibility is a direct application of both the extensible part

of XML and the Darwin element in DITA: XML elements can be extended an DITA information types can evolve to accommodate diverse content and pro cessing needs.

DITA Maps

Properly tagged DITA topics can create or join content collections ready fo reuse of material, single-sourcing of whole topics or their elements, and filter as determined by a deliverable's audience or context. The process of assemblin topic collections depends on DITA maps, which are defined as "the glue tha binds your topics together, the driver for producing your output, and the infor mation path for your users to follow" (Bellamy et al., 2012, p. 91).

In their textbook *DITA Best Practices*, Bellamy et al. recommend using DIT maps to create an information set that specify which topics should be include in a user deliverable produced from the map, define an information architectur with the navigation for a set of topics, and create relationships between topic (Bellamy et al., 2012).

The following scenario combines the examples I have presented so far: Che Pedro has a website in which he critiques recipes. For this week's entry, he wi critique the marinara sauce recipe from Figure 1.3. The web deliverable tha Chef Pedro needs for this project should include the following sections:

- The marinara sauce recipe (structured as a DITA task)
- Chef Pedro's critique of the recipe (structured as a DITA concept)
- Chef Pedro's "about" page, with biographical and professional informatio (structured as a DITA concept).

Figure 1.6 shows a very basic DITA map for Chef Pedro's recipe critique websit

The DITA map has a **<title>** element for the whole project. Then, the ma includes links (**<topicref>**) for the individual files needed for the website. Usin the DITA-OT, which I will cover in more detail in Chapters 5 and 6, I transforme

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
   <title>Chef Pedro's Recipe Critique</title>
   <topicref href="t-marinara.dita" />
   <topicref href="c-marinara-critique.dita" />
   <topicref href="c-about.dita" />
</map>
```

FIGURE 1.6 DITA map for Chef Pedro's recipe critique website. The map include links (or topic references) to the marinara sauce recipe, Chef Pedro's critique of the recipe, and an "about" page with information about the author.

the map to a web deliverable. The DITA-OT automatically generated a navigation menu (figure 1.7) and took care of basic layout.

These features and benefits contribute to DITA's popularity and adaptability as an intelligent content solution for technical communication. Without a doubt, these features and benefits can also be intimidating for an author new to DITA. Although no author is expected to become an expert on all topic types and their relations, criticism in industry and, perhaps, the timid interest from academia stem from the long, and getting longer, list of types, tags, and their functions included in the DITA standard.

Let's go back to Chef Pedro, who is attempting now to structure some of his own recipes as tasks following specifications from the DITA 1.3 standard. The process sounds simple, and the content components in the DITA *task* document type seem just right for structuring a recipe. There can be some instances of the **<step>** element with a **<cmd>** for a command in each step, but things get quite complicated when Pedro looks at all the available elements in a task topic[2], which include the following: **<taskbody>**, **<prereq>**, **<context>**, **<steps>**, **<steps-informal>**, **<steps-unordered>**, **<step>**, **<stepsection>**, **<cmd>**, **<info>**, **<substeps>**, **<substep>**, **<stepxmp>**, **<choicetable>**, **<chhead>**, **<choptionhd>**, **<chdeschd>**, **<chrow>**, **<choption>**, **<chdesc>**, **<choices>**, **<steptroubleshooting>**, **<stepresult>**, **<tutorialinfo>**, **<tasktroubleshooting>**, **<result>**, and **<postreq>**. And that's just the task document type! Keep in mind that DITA 1.3 has 26 document types.

The next section introduces a DITA-based approach to intelligent content that has the potential of minimizing the standard's learning curve and promoting its adoption in small and medium scale information authoring and processing environments as well as classrooms.

FIGURE 1.7 Navigation menu for a web transformation of Chef Pedro's recipe critique project. The DITA-OT generated the menu and took care of basic layout for this web deliverable.

DITA, Why Go Lightweight?

Michael Priestley, Senior Technical Staff Member and Enterprise Conter Technology Strategist at IBM and known as one of DITA's "founding fathers (Etheridge, 2016), defines Lightweight DITA (LwDITA, which should be rea as "Lightweight DITA") as a simplified schema for structuring content, wit fewer elements, tighter content models and a simplified specialization archi tecture to define new types compared to those of DITA XML. According t Priestley, "if there are three ways of doing things with full DITA, there will b only one way to do it with Lightweight DITA."

> That simplification makes it possible to implement DITA without XML for example using Markdown, or HTML5. This brings the advantages c structured authoring to where people are already creating content, rathe than trying to get every author onto one content platform. Lightweigh DITA can be the glue that ties together many different authoring plat forms across a company. It can also be an on-boarding ramp for fu DITA. Lightweight DITA is particularly attractive to companies wh need a faster ROI [return on investment] and an easier learning curve.
>
> (Priestley, quoted in Etheridge, 201

Creating Intelligent Content with Lightweight DITA introduces and ana lyzes LwDITA as an approach for developing intelligent content. It aims t address concerns and doubts about the adoption and evolution of DITA as major standard for technical publications. Those concerns and doubts revolv around a major point of discrepancy in the world of technical communicatior in industry, XML and DITA are widely used and some critics even see them a outdated and complex; in academia, "faculty are simply ignoring the subjec even though it has played a central role in the practitioner literature" (Clarl 2016, p. 19). In conversations at academic conferences and in social medi exchanges, colleagues from other universities tell me they would like to tr DITA in the future and make plans to contact me when they are working o their technical communication syllabus. I rarely get a second call and the cor versation stays in the "someday/maybe" list. If XML and DITA are still seen a new tools by academics when some practitioners are already labeling them a arcane, then we are just writing a new chapter in the documented gap betwee research and practice in the field (Andersen, 2013; Rude, 2015; Lauren & Pigg 2016, and others).

For an audience of curious but hesitant academics, this book demystifies th process of authoring and processing intelligent content. *Creating Intelliger Content with Lightweight DITA* shows structured authoring in practice with out the need for sophisticated software tools. It also connects the rhetoric

structures that drive the evolution of DITA as a standard for technical information to foundational concepts from the field's academic background and to common structures in computing education. For interested practitioners, this book presents the logical and computational essence of changes and improvements on a major documentation standard. For any reader, *Creating Intelligent Content with Lightweight DITA* places human authors at the center of intelligent content workflows, proposing that adopting LwDITA as a thought process and authoring schema will provide writers with some of the benefits associated with other information standards (DITA included) while **emphasizing thinking and abstraction over automation and machine processing**. This book is not about a specific tool or software platform: as an open standard in development, LwDITA does not depend on a vendor or "app." Hackos explains the importance of information standards for technical communicators with the following scenario, which also applies to LwDITA:

> What that means to information developers is that you can author a DocBook or DITA topic in one tool and open the topic in another tool without the loss of information or invalid markup. If the tool developers respect the standard, they allow for interoperability among tools.
>
> (Hackos, 2016, p. 29)

Based on the LwDITA proposed standard, the principles and strategies included in this book can be effective in a variety of authoring environments. I am aware, though, that some large content repositories with diverse user groups do need a robust environment built on DITA XML, and that strong component of the standard is not going away. I am a voting member of the DITA Technical Committee with the Organization for the Adoption of Structured Information Standards (OASIS), which is currently working on version 2.0 of the standard; DITA's present and future are necessary for the lightweight approaches discussed in this book. Therefore, this is unequivocally a book about DITA, but it is not a detailed introduction and how-to to the widely adopted XML-based DITA standard (for comprehensive introductory texts to DITA XML, see Hackos, 2011; Bellamy et al., 2012).

Creating Intelligent Content with Lightweight DITA does not have a reader prerequisite of experience with the DITA standard. However, I do assume that readers from industry and academia will have a technological curiosity about content development in workflows that go beyond a word processor or a "What You See Is What You Get" (WYSIWYG) web editor. Some experience with HTML or Markdown would also help, but the book provides enough context and information about those languages to understand and use the LwDITA proposed standard.

Computational Thinking and the Evolution of DITA

The theoretical axis of the evolution of DITA analyzed in this book is based o the concept of computational thinking, which is defined by their main propc nents as follows:

> Computational thinking is the thought processes involved in formulatin problems and their solutions so that the solutions are represented in a forr that can be effectively carried out by an information-processing agent.
> (Cuny et al. 2010, quoted by Wing, 2011

Computational thinking (CT) has been connected in scholarship and medi to initiatives for learning to code and programming. In her seminal essay on th topic, Wing explains that computational thinking takes an approach to "solvin problems, designing systems and understanding human behavior by drawin on concepts fundamental to computer science" (Wing, 2006, p. 33), and that is more about conceptualizing than programming. She adds that "thinking lik a computer scientist means more than being able to program a computer. requires thinking at multiple levels of abstraction" (2006, p. 35). She succinctl defines abstraction as "the essence of computational thinking" (2008, p. 3717)

In the technical communication literature, the concept of abstraction ha been linked to Johndan Johnson-Eilola's work to establish "common groun between academic and corporate models" of the profession. Johnson-Eilola skills for rearticulating technical communication included abstraction, whic "requires students not merely to memorize information but also to learn t discern patterns, relationships, and hierarchies in large masses of informa tion" (Johnson-Eilola, 1996, p. 260). Johnson-Eilola adds that a "paradigmati example of this skill can be found in one of the most common tasks in softwar documentation: rethinking a series of system commands so that it coincide with a user's task representation and context" (1996, p. 260). More than tw decades after Johnson-Eilola's model was published, abstraction continue to be a desired skill in the training and work of technical communicator Furthermore, his example is still relevant when thinking about LwDITA an computational thinking: an author needs to separate the layers of abstractio when creating a task topic for a specific audience and context. The tools migl be different, but the principles are the same.

Abstraction allows authors of intelligent content to separate the "layers" of particular problem, and work on each one individually without concern for th others. Then, as the layers are recombined, they work together to solve the prob lem at hand, much like an algorithm. In the case of a computational solution fc intelligent content, abstraction is a combination of two elements: informatio representation and separation of concerns (layers of abstraction). Computing concerned with automating these abstractions. In order for that automation t

be successful, computational thinking requires understanding not only of the concepts that each layer of abstraction represents, but also of the relationships between the multiple layers behind a specific problem.

Some computing professionals will argue that the abstractions in computational thinking are mainly related to algorithms. Some technical communication practitioners share that opinion (Baker, 2016). In Wing's model, however, computing abstractions go beyond numerical abstractions and cover symbolic, algorithmic, and representational abstractions (Wing, 2008). Others will emphasize the role of automation even in Wing's definition of computational thinking. *Creating Intelligent Content with Lightweight DITA* does not have the purpose of minimizing the importance of computing automations, and by all means technical communicators should learn computing programming languages and workflows if given the chance. This book treats ambitious promises of computational thinking with caution, as "there is little evidence to believe that students are learning higher-order thinking skills by learning programming" (Guzdial, 2016, p. 50). Nonetheless, thinking in and planning abstractions are really the everyday tasks of technical authors in an intelligent content environment. Wing (2008) describes computing as a combination of "mental" tools (abstractions) and "metal" tools (automation). Authors are in charge of the abstractions behind the code and content that provides the backbone for intelligent content. Automation is then provided by software applications, be it a commercial product like Adobe FrameMaker or an open source implementation of DITA. Authors do not need special software tools to create content in DITA; however, they will need a software processor to transform DITA topics into deliverables for human users. I will describe those processing applications later in this chapter (and throughout the book). Authors can participate in automation by writing a script or application to filter content. However, that step is not necessary for applying principles of computational thinking in technical communication from an author's perspective. I have seen colleagues in technical communication who value computational thinking and literacy as core skills of a college education and, as a result, send their students to take an introductory class in programming in a Computer Science department. Before (or in addition to) taking, say, a Python course out of their disciplinary context, why not send technical communication students to a course covering the *mental* and *metal* tools that practitioners in their field value?

A core argument in this book is based on Wing's definition of computational thinking, previous research about abstractions in technical communication, the work I have conducted as co-chair of the Lightweight DITA subcommittee at OASIS, and my experience teaching DITA at the college level since 2006 and LwDITA since 2015. I argue that **if technical communicators are trained with principles of rhetorical problem solving and computational thinking, they can work in lightweight environments without the need of a robust XML solution**. This type of training will primarily enhance an author's understanding

of the layers of abstraction and common argumentation moves behind an inte ligent content solution. These layers of abstraction are not necessarily new t the field of technical communication (they include long-discussed concepts an principles like separating content from design, and planning for single-sourcin and content reuse, among others), but are essential building blocks of intelliger content solutions that are still not fully embraced in academia. Before LwDIT existed, I presented a preliminary list of those abstractions as they applied to m teaching of DITA XML (Evia et al., 2015). Chapters 7 and 8 present an analys of the abstractions behind the computational thinking and rhetorical proble solving processes that generate intelligent content with LwDITA, but the ne section focuses on the potential benefits of LwDITA for content developers wh do not need the features of full DITA XML.

Structuring Intelligent Content with LwDITA

LwDITA is a topic-based architecture for tagging and structuring intelliger content using flexible markup options. Lightweight DITA aims to streamlin the DITA authoring experience by presenting three formats for content creatio

- *XDITA*, an XML format with a subset of DITA elements that can be use for validated authoring and complex publishing chains
- *HDITA*, an HTML5 format that can be used for either authoring c displaying content
- *MDITA*, a Markdown[3] format with a subset of XDITA elements that can b used for maximizing input readability while maintaining structure in conten

You do not need to use all three "flavors" at the same time to adopt LwDIT You can work in HDITA all the time and you would still be using LwDIT You can live in an MDITA environment without XML or HTML tags and yo would still be using LwDITA. All three LwDITA formats are compatible wit each other and with DITA XML. For a team of authors with diverse techn cal backgrounds and communication skills, the different formats of LwDIT allow collaboration and content exchange in a centralized solution. For exam ple, Pedro can hire a technical writer to create recipe topics in XDITA (base on XML) while a marketing professional writes a description of the cookbook features in HDITA (based on HTML5), and an engineer uses MDITA (base on Markdown) to create a reference for a specific command from the kitchen laboratory. All their topics are treated as DITA and can take advantage of th standard's reuse, filtering, and single-sourcing capabilities.

All code examples in this book will focus on the open standards for DIT and LwDITA, and automation-related discussions will be based on features an affordances from the open source DITA-OT and "raw" XML, HTML5, an Markdown code. Professional authoring tools can hide code in What You Se

Is What You Get (WYSIWYG) options, but the rhetorical, pedagogical, and computational principles of *Creating Intelligent Content with Lightweight DITA* view and perceive LwDITA topics as code.

The idea of simplified DITA code that would reduce its learning curve has been a topic of discussion on the standard's technical committee with OASIS for a few years. In 2011, the technical committee planned to release a "limited DITA profile," which was still XML-based, but depended heavily on HTML tags (such as **<p>** for paragraph and **<li>** for list item) to simplify many semantic structures of DITA XML. As the concept of Lightweight DITA developed further, at one point it became an XML subset of DITA that included, for example, 27 possible elements inside a topic, whereas DITA XML includes a possible combination of 90+ elements. Originally, Lightweight DITA was planned as a component of the DITA 1.3 specification, but interest from members of the DITA technical committee, vendors, and researchers pushed it out of the main specification and into its own parallel and compatible standard. The purpose of LwDITA is not to replace DITA XML. Instead, LwDITA provides basic access to authors who do not need all the DITA standard's features but whose deliverables should be compatible with DITA XML.

Michael Priestley, from IBM, created the OASIS Lightweight DITA subcommittee in 2014 with the purpose of releasing LwDITA as an open standard related to but independent from DITA XML. I co-chair the Lightweight DITA subcommittee with Priestley, and I have published, alone and with Priestley, about the development of LwDITA (e.g., Evia & Priestley, 2016; Evia, 2017). While LwDITA is under development and not an approved OASIS standard at the time of this writing, feedback on our talks and papers about the new standard provides support for its development based on positive reactions and interest from partners in industry and academia. As lead editor of the LwDITA technical specification, I have attempted to combine the needs and resources of academic and industry content professionals, testing and implementing applied computational principles built on common concepts of genre and rhetorical theory.

As of this writing, LwDITA is a work in progress. This book reflects the structure of this proposed standard as it was presented in its initial introductory committee note (Evia et al., 2018). LwDITA details might change between the publication of this book and the actual release of the Lightweight DITA standard.

For a quick example of LwDITA in action (and I will analyze *more thoroughly* its formats later in the book), I coded O'Keefe's recipe for marinara sauce in LwDITA's different authoring formats. Figure 1.8 shows the recipe authored in XDITA – the LwDITA authoring format based on a simplified version of DITA XML.

The first major change is the topic type. In DITA XML (see Figure 1.3), the recipe was structured as a task and had predetermined elements that the DITA standard associates with a task (like **<steps>** and **<prereq>**). The simplified authoring experience of LwDITA, however, is based on a single topic type with elements common to most information units, including the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD LIGHTWEIGHT DITA Topic//EN"
"lw-topic.dtd">
<topic id="t-marinara">
    <title>Marinara sauce</title>
    <shortdesc>Prepare a crowd-
pleasing red sauce for pasta in about 30 minutes.</shortdesc>
    <prolog>
        <data name="author" value="Unknown"/>
        <data name="category" value="Italian"/>
    </prolog>
    <body>
        <section>
            <title>Ingredients</title>
            <ul>
                <li>
                    <p>2 tbsp. of olive oil</p>
                </li>
                <li>
                    <p>2 cloves of garlic, minced</p>
                </li>
                <li>
                    <p>1/2 tsp. of hot red pepper</p>
                </li>
                <li>
                    <p>28 oz. of canned tomatoes, preferably San Marzano</p>
                </li>
                <li>
                    <p>2 tbsp. of parsley, chopped</p>
                </li>
            </ul>
        </section>
        <section>
            <title>Preparation</title>
            <ol>
                <li>
                    <p>Heat olive oil in a large saucepan on medium</p>
                </li>
                <li>
                    <p>Add garlic and hot red pepper and sweat until fragrant
</p>
                </li>
                <li>
                    <p>Add tomatoes, breaking up into smaller pieces</p>
                </li>
                <li>
                    <p>Simmer on medium-low heat for at least 20 minutes</p>
                </li>
                <li>
                    <p>Add parsley</p>
                 </li>
                 <li>
                   <p>Simmer for another five minutes</p>
                 </li>
                 <li>
                    <p>Serve over long pasta.</p>
                 </li>
             </ol>
        </section>
    </body>
</topic>
```

FIGURE 1.8 Marinara sauce recipe as an XDITA topic. The XML tags are still visib and the recipe looks like a DITA topic. However, it is no longer a task since LwDITA's initial specification only includes one topic type.

- Title: A label that connotes the purpose of the content that is associated with it
- Short description: A brief depiction of the purpose or theme of a topic
- Prolog: A container for metadata about a topic (for example, author information or subject category)
- Body: A container for the main content of a topic. It might include several sections
- Section: An organizational division within a topic. It can have an optional title.

These common elements can be represented, with modifications to accommodate different authoring languages, in XDITA, HDITA, and MDITA. Figure 1.9 shows the same recipe as a topic tagged in HDITA, the LwDITA authoring format that uses HTML5.

The topic has commonly-used HTML5 elements like headings and lists, but an additional benefit of HDITA is that topics authored in this LwDITA format do not require a transformation process to generate a publishable outcome (Figure 1.10). Because the topics are HTML5 files, they can be rendered in any web browser, and publishers can customize the rendered format with a standard CSS stylesheet.

MDITA, the LwDITA authoring format that uses Markdown, can also be used to structure the recipe for marinara sauce (figure 1.11). In its core profile, MDITA provides structure for major elements present in DITA XML and Markdown (title, sections, lists, etc.). In its extended profile, MDITA allows a header authored in YAML[4] (the recursive acronym for *YAML Ain't Markup Language*) with metadata about the topic's author and some categories that we have been carrying since the original XML sample.

Regardless of its authoring format, when transformed with LwDITA-aware tools into information deliverables for human users, the topic for the marinara sauce recipe would pretty much look the same. That is a key feature of LwDITA: end users will not know the author's process and will just receive information products with consistent structure. Figure 1.12 shows a PDF version of the XDITA topic created with the DITA-OT.

An author trained in principles of structured authoring could use XDITA and take advantage of its DITA-like sections, elements, and constraints. XDITA provides some of the DITA mechanisms for reuse and single sourcing that could be essential for a technical communicator but probably distracting or confusing for a casual content contributor. HDITA can be less intimidating for collaborators with experience creating content in HTML5, while still including reuse and filtering options. Both XDITA and HDITA can be authored in WYSIWYG editors that keep code and tags hidden (permanently or temporarily, depending on the specific software tool) from the content creator. MDITA is a plain text variant for developers and authors who do not need advanced content reuse capabilities (but they still can use them with raw HDITA code fragments). All three formats, however, are compatible with each other and also with topics created according to the DITA XML standard. All three formats also incorporate fundamental actions of content authoring, like staging, coaching, and describing, which are essential moves of

```
<!DOCTYPE html>
<meta name="author" content="Unknown">
<meta name="keywords" content="Italian">
       <title>Marinara sauce</title>
      <body>
        <article id="t-marinara">
            <h1>Marinara sauce</h1>
            <p>Prepare a crowd-
pleasing red sauce for pasta in about 30 minutes.</p>
            <h2>Ingredients</h2>
            <ul>
                <li>
                    <p>2 tbsp. of olive oil</p>
                </li>
                <li>
                    <p>2 cloves of garlic, minced</p>
                </li>
                <li>
                    <p>1/2 tsp. of hot red pepper</p>
                </li>
                <li>
                    <p>28 oz. of canned tomatoes, preferably San Marzano</p>
                </li>
                <li>
                    <p>2 tbsp. of parsley, chopped</p>
                </li>
            </ul>
            <h2>Preparation</h2>
            <ol>
                <li>
                    <p>Heat olive oil in a large saucepan on medium</p>
                </li>
                <li>
                    <p>Add garlic and hot red pepper and sweat until fragrant
</p>
                </li>
                <li>
                    <p>Add tomatoes, breaking up into smaller pieces</p>
                </li>
                <li>
                    <p>Simmer on medium-low heat for at least 20 minutes</p>
                </li>
                <li>
                    <p>Add parsley</p>
                </li>
                <li>
                    <p>Simmer for another five minutes</p>
                </li>
                <li>
                    <p>Serve over long pasta.</p>
                </li>
            </ol>
         </article>
   </body>
```

FIGURE 1.9 Marinara sauce recipe as an HDITA topic. The topic is now an HTML5 article, and the elements' structure looks very similar to what can be accomplished with XML.

Marinara sauce

Prepare a crowd-pleasing red sauce for pasta in about 30 minutes.

Ingredients

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

Preparation

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

FIGURE 1.10 HDITA recipe for marinara sauce seen on a web browser. An added benefit of HDITA is an instant presentation view that does not require processing or transformation to generate a basic publishable outcome.

```
---
id: t-marinara
author: Unknown
category: Italian
---
# Marinara Sauce
Prepare a crowd-pleasing red sauce for pasta in about 30 minutes.

## Ingredients
- 2 tbsp. of olive oil
- 2 cloves of garlic, minced
- 1/2 tsp. of hot red pepper
- 28 oz. of canned tomatoes, preferably San Marzano
- 2 tbsp. of parsley, chopped.

## Preparation
1. Heat olive oil in a large saucepan on medium
2. Add garlic and hot red pepper and sweat until fragrant
3. Add tomatoes, breaking up into smaller pieces
4. Simmer on medium-low heat for at least 20 minutes
5. Add parsley
6. Simmer for another five minutes
7. Serve over long pasta.
```

FIGURE 1.11 Marinara sauce recipe as an MDITA topic. The major change in this version is the absence of tags to represent elements.

technical communication (Eli Review, n.d.) that act as commonplace elements i the repertoire of an author. Chapter 3 will look at how those common moves c technical communication relate to content structures in DITA and LwDITA.

For processing purposes, a single DITA map can combine topics created i different LwDITA formats (Figure 1.13).

Deliverables created from the sample map in Figure 1.13 can include a prir cookbook or an online recipe guide, based on the automation tools used by Ch Pedro and his team in a specific publishing scenario.

Computer Code for Human Authors

DITA and all three LwDITA formats are undeniably code. Calling them "com puter code," however, could offend programmers and developers. Particularl for technical communication students and practitioners with backgrounds i writing and the Humanities, *this is* their computer code. They will probably nc use advanced programming languages, but they work with XML, HTML5, an even Markdown code that for them requires a different kind of thinking tha desktop publishing workflows involving long document files with a word prc cessor. These are the skills that someone like IBM's James Mathewson expec from a technical communication graduate.

Authors using any combination of LwDITA formats do not need new tec nological skills. They will continue "the move away from a document-based to

Marinara sauce

Prepare a crowd-pleasing red sauce for pasta in about 30 minutes.

Ingredients

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

Preparation

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

FIGURE 1.12 PDF version of the XDITA topic for the marinara sauce recipe. Compare to the deliverable produced from a DITA XML task in Figure 1.4. The topic gained sub-headings because the <section> element replaced more specific moves associated with a task.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
    <title>Fantastic Cookbook</title>
    <topichead>
    <topicmeta><navtitle>Sauces and condiments</navtitle></topicmeta>
    <topicref href="t-tikkamasala.dita" format="dita" />
    <topicref href="t-mole.html" format="hdita" />
    <topicref href="t-marinara.md" format="mdita" />
    </topichead>
</map>
```

FIGURE 1.13 DITA map aggregating different LwDITA formats. The end users will see all the recipes with the same structure regardless of authoring process.

topic-based approach to developing, managing, and publishing content" (Andersen 2014, p. 116) widely adopted in the field of technical communication. New knowledge will actually come in the form of abstractions that allow authors to identify rhetorical moves (e.g., staging with a **<shortdesc>**, guiding with **<steps>**, showing with **<example>**) that were strictly enforced in DITA XML but are not required in LwDITA. In DITA, for example, an author has strict tags for a short description element and hazard statements, whereas in some LwDITA formats those sections are just paragraphs. These thought processes follow the foundations of the skill set labeled as computational thinking in recent literature about computing education.

LwDITA is not for everyone. In a large organization with limited resources to train authors, the content integrity and structural consistency provided by DITA XML might be the best solution. Fortunately, DITA is still evolving and its technical committee at OASIS is at the time of this writing planning version 2.0 of this content standard. LwDITA does not have the objective of replacing DITA which is still available for authors and teams who need its full capabilities. The abstraction tasks from this framework, however, will enable critical users to apply computational thinking and technical communication principles in a series of layers that reveal intelligent content structure beyond what a software tool allows.

The 2012 Adobe video described a future of technical communication that now is more like its present, or even its past. For some, technical communication is still about structured information and intelligent content that adapts to users' needs. However, complex XML code is only one way (and maybe approaching obsolescence) to get there. Simplified lightweight markup (or *markdown*) cannot be ignored because authors are creating topics directly on the online environments where they will be read; after all, not every publishing project needs advanced reuse and filtering. Furthermore, coding and automating content delivery is but a "metal" element in the complicated process behind authoring and publishing intelligent content. Human beings in charge of content creation need to move their "mental" focus to the abstraction thinking behind the rhetorical decisions that make content intelligent.

Before we move on to specific how-to and examples of creating intelligent content with LwDITA following principles of computational thinking, we need to revisit the origins of DITA XML and analyze how its content structures and discourse conventions (based on the archetypal computer manual) evolved into LwDITA, and those are the main themes of the next chapter.

Notes

1 I present the recipe in the source's original Pascal Case; all other code examples in this book will use lowercase, which is more commonly associated with DITA best practice
2 http://docs.oasis-open.org/dita/dita/v1.3/errata01/os/complete/part2-tech-content/langRef/containers/task-elements.html#task2
3 "a plain text format for writing structured documents, based on formatting conventions from email and usenet," http://commonmark.org
4 http://yaml.org

References

2.2.1.1 The topic as the basic unit of information. (2016, October 25). Retrieved from http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/archSpec/base/topicdefined.html#topicdefined

2.7.1.6 Troubleshooting topic. (2016, October 25). Retrieved from http://docs.oasis-open.org/dita/dita/v1.3/errata01/os/complete/part3-all-inclusive/archSpec/technicalContent/dita-troubleshooting-topic.html#dita-troubleshooting-topic

2.7.1.7 Glossary entry topic. (2016, October 25). Retrieved from http://docs.oasis-open.org/dita/dita/v1.3/errata01/os/complete/part3-all-inclusive/archSpec/technicalContent/dita-glossary-topic.html#glossaryArch

AdobeTCS. (2012, July 16). *Future of TechComm.* [Video File]. Retrieved from https://youtu.be/dSdhnyDF0YY

Andersen, R. (2013). The value of a reciprocal relationship between research and practice. *Information Management News*. Retrieved from http://www.infomanagementcenter.com/publications/e-newsletter/may-2013/the-value-of-a-reciprocal-relationship-between-research-and-practice/

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.

Andersen, R., & Batova, T. (2015). The current state of component content management: An integrative literature review. *IEEE Transactions on Professional Communication*, 58(3), 247–270.

Applen, J. D., & McDaniel, R. (2009). *The rhetorical nature of XML: Constructing knowledge in networked environments.* New York: Routledge.

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.

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

Baker, M. (2016, March 1). Algorithms: Separating content from formatting. Retrieved from https://techwhirl.com/algorithms-separating-content-from-formatting/

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.

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

Cowan, C. (2010). *XML in technical communication* (2nd ed.). Peterborough: Institute of Scientific and Technical Communication.

Day, D., Priestley, M., & Schell, D. (2001, March 1). *Introduction to the Darwin Information Typing Architecture.* Retrieved from http://www.ibm.com/developerworks/library/x-dita1/

DITAWriter (n.d.). Companies Using DITA. Retrieved June 19, 2018, from http://www.ditawriter.com/companies-using-dita/

DITAWriter. (2016, July 15). *Don Day and Michael Priestly [sic]* on the beginnings of DITA: Part 2. Retrieved from http://www.ditawriter.com/don-day-and-michael-priestly-on-the-beginnings-of-dita-part-2/

Eberlein, K.J. (2016). Content reuse. In R. Gallon (Ed.) *The Language of Technical Communication* (pp. 54–55). Laguna Hills, CA: XML Press.

Eli Review. (n.d.). The Essential moves of technical communication. Retrieved fro
http://elireview.com/content/curriculum/techcom/

Etheridge, A. (2016, May 17). Experts talk DITA & localization – Michael Priestley. Retrieve
from http://www.whp.net/en/experts-talk-dita-localization-michael-priestley/

Evia, C. (2017). Authoring standards-based intelligent content the easy way wit
Lightweight DITA. *Proceedings of the 35th ACM International Conference on th
Design of Communication.*

Evia, C., Sharp, M. R., & Perez-Quiñones, M. A. (2015). Teaching structured autho
ing and DITA through rhetorical and computational thinking. *IEEE Transactions o
Professional Communication*, 58(3), 328–343.

Evia, C., & Priestley, M. (2016). Structured authoring without XML: Evaluating ligh
weight DITA for technical documentation. *Technical Communication*, 63(1), 23–37

Evia, C., Eberlein, K., & Houser, A. (2018). *Lightweight DITA: An introduction.* Versio
1.0. OASIS.

Gesteland McShane, B. (2009). Why we should teach XML: An argument for technic
acuity. In G. Pullman & B. Gu (Eds.) *Content management: Bridging the gap betwee
theory and Practice* (pp. 73–85). Amityville, NY: Baywood Pub.

Glushko, R. J., & McGrath, T. (2008). Document engineering: Analyzing and designin
documents for business informatics and web services. Cambridge, MA: The MI
Press.

Guzdial, M. (2016). *Learner-centered design of computing education: Research on con
puting for everyone.* San Rafael, CA: Morgan & Claypool.

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

Hackos, J. T. (2016). International standards for information development and conte
management. *IEEE Transactions on Professional Communication*, 59(1), 24–36.

Johnson-Eilola, J. (1996). Relocating the value of work: Technical communication in
post-industrial age. *Technical Communication Quarterly*, 5(3), 245–270.

Kaplan, N. (2014, May 3). *The death of technical writing, part 1.* Retrieved from https:
customersandcontent.com/2014/05/03/the-death-of-technical-writing-part-1/

Kerzreho, N. (2016). Component content management system. In R. Gallon (Ed.) *Th
language of technical communication* (pp. 60–61). Laguna Hills, CA: XML Press.

Kimber, E. (2012). *DITA for practitioners, volume 1: Architecture and technology.* Lagun
Hills, CA: XML Press.

Lauren, B., & Pigg, S. (2016). Toward multidirectional knowledge flows: Lessons fro
research and publication practices of technical communication entrepreneur
Technical Communication, 63(4), 299–313.

Mathewson, J. [James_Mathewson]. (2016, September 8). Yes. But writing reusab
content is a rare skill. Would that they taught it in college. Essays from whole clot
is more the thing. [Tweet]. Retrieved from https://twitter.com/James_Mathewso
status/774044623840870400

O'Keefe, S. (2009). Structured authoring and XML. In O'Keefe, S. (Ed.) *The compas
Essential reading about XML, DITA, and Web 2.0.* Durham, NC: Scriptoriu
Publishing Services, Inc.

OASIS Darwin Information Typing Architecture (DITA) TC. (n.d.). Retrieved fro
https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita

Priestley, M., Hargis, G., & Carpenter, S. (2001). DITA: An XML-based technical documentation authoring and publishing architecture. *Technical Communication*, 48(3), 352–367.

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

Rockley, A., Cooper, C., & Abel, S. (2015). *Intelligent Content: A Primer*. Laguna Hills, CA XML Press.

Rude, C. D. (2015). Building identity and community through research. *Journal of Technical Writing and Communication*, 45(4), 366–380.

Sapienza, F. (2002). Does being technical matter? XML, single source, and technical communication. *Journal of Technical Writing and Communication*, 32(2), 155–170.

Stolley, K. (2008). The lo-fi manifesto. *Kairos: A Journal of Rhetoric, Technology, and Pedagogy* 12(3). Retrieved from http://kairos.technorhetoric.net/12.3/

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

White, L.W. (2016). Single sourcing. In R. Gallon (Ed.) *The language of technical communication* (pp. 56–57). Laguna Hills, CA: XML Press.

Wing, J. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Wing, J.M. (2011). Research notebook: Computational thinking – what and why? Retrieved from https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

2

BEFORE INTELLIGENT CONTENT, THERE WAS THE COMPUTER MANUAL

John M. Carroll is held responsible for some actions and decisions that hav defined the academic and professional field of technical communication as w know it today. Carroll's work on applying principles of minimalism to techn cal documentation, presented chiefly in his book *The Nurnberg Funnel* (1990 still resonates with professionals advancing theory and practice related t procedural writing, delivery of instruction, and interface design. At the sam time, and in a more tongue-in-cheek fashion, Carroll has also been labeled a "the man who killed the manual" (Svenvold, 2015). A casualty in the manual demise, one could argue, was the main line of research that connected prac titioners and academics in technical communication. As a proposed standar for authoring and publishing content that includes technical communicatio Lightweight DITA shares a common ancestor with the computer manual of th past. Therefore, LwDITA can be described as a resource to mend some aspec of the research-to-practice loop in our discipline.

The title of Carroll's influential book refers "to the legendary Funnel c Nurnberg, said to make people wise very quickly" (1990, p. 10). Carroll used th term to describe approaches to self-instruction related to computing that expecte that a user would consume content at their own pace and then come out com petent, after returning to "varying levels of incompetence" (p. 4) in the use of specific hardware or software system. Carroll's proposed approach was a commit ment to "minimizing the obtrusiveness to the learner of training material — henc the term *minimalist*" (p.7, his emphasis). The three key aspects of Carroll's min malist approach to instruction are the following:

1. Allowing learners to start immediately on meaningfully realistic tasks
2. Reducing the amount of reading and other passive activity in training

3. Helping to make errors and error recovery less traumatic and more pedagogically productive (p. 7).

In Svenvold's account of the events, the manual was doomed when Carroll applied those principles of minimalism to the production of that specific genre of technical communication.

> Short, succinct manuals allow the user to dive into many different tasks and to accomplish them quickly, thereby gaining a sense of control and autonomy that inspires further learning.
>
> (Svenvold, 2015)

The "death" of the manual as the default genre of technical communication has impacted the field in more than one way. Gone are the days when the manual was binding the academic and practitioner camps of technical communication as they worked towards the advancement of this common genre. At professional conferences, I occasionally engage in nostalgic conversations with colleagues who continue to state the claim that content developers *and their readers* lost something when the manual disappeared. A certain kind of colleague who still writes in a command-line environment using a text editor like Vim would promptly procure pseudo-evidence to build a case arguing that developers and users have lost the "itch" to tinker with software and hardware because the manual is no longer there to guide them and provide hundreds of reference points. That same colleague would also point out examples of independent software companies that still produce lengthy user manuals.

Except they did not. In John Carroll's opinion, today's developers and users have not really lost the itch to tinker with systems because the manuals disappeared. "A lot of things changed," Carroll said. "One thing is that back in the day, people had to be tinkerers because things didn't work and they were clunky and you had to know your Unix and not be scared of the command line blinking. We live in a much better world." Carroll pointed out that those early days of computing were dominated by data-processing professionals in a highly selective environment: "They were mostly white men, and now computers are used by everybody in the world" (J. Carroll, personal interview, 2015, May 28).

It was a warm day in May when I interviewed John Carroll at his office on the Penn State campus, and he looked more like a Jimmy Buffett concertgoer than the patron saint of technical communicators. We reminisced about the Virginia Tech Center for Human-Computer Interaction (CHCI), which Carroll founded in 1995. As I write this, I am a member of the CHCI executive committee and chair of its education-oriented strategic planning group. Yet, John Carroll and I never worked together, as he left Virginia Tech a few months before I arrived as an assistant professor in 2004. I insisted on the potential consequences of losing the flagship genre of technical communication, and Carroll painted a bigger picture:

"It's not just the manual, but the world has changed, and the software and docu mentation business model. We cannot afford the manual. Actually, the answer (fo the manual's disappearance) is not minimalism, but money." Carroll explaine that with computing systems like notebook computers and smartphones (whic are considerably more affordable than their mainframe ancestors in the 1980s an earlier), "there's no way to economically produce definitive manuals that chea There's no business model for that."

Carroll admitted that, at a personal level, he shares the feeling of nostalgia ov the missing manuals. "When I open a box, and I take something out and there's n manual, that's my first thought: Where's the manual?" For a moment, he acknow edged that readers lost something when the manual became an extinct genre: "Th instruction manual was also a symbol of empowerment, almost democratic; that another source of the nostalgia. If I have a manual, I am as good as the expert: have the information and if I don't I have to call a help number and admit tha someone else is in control." But then he hammered in the last nail in the manual coffin: "All this being said . . . the manuals didn't work. People didn't read then Even today, they still have the plastic on them in libraries. Nobody read them."

John Carroll Takes the Stand

We then moved on to the technical communication-related accusations tha hang over John Carroll's head. How does Carroll plead when facing the accusa tion of killing the manual?

> I find it very flattering that anyone can think I can have that effect o anything. I honestly think it was much more a matter of a paradigm shif a change in business model, changing populations, and the fact that th computer became a universal device instead of an elite device. Those wer forces bigger than minimalism. I don't think an information model did i
>
> (Carroll, personal interview, 2015, May 2

Carroll added that the death of the manual is not the only factor responsibl for the disconnect in interests and priorities between academia and industr in technical communication. He described that as a problem of scaling: whe there were a handful of important tech companies, their research and develop ment departments could work closely with academics, and those partnership were reciprocal. Professors would benefit from knowledge applied in thei courses and research projects, and practitioners operated under the idea tha professors had something important to tell. Carroll identified the same pa tern of disparities in computing-related fields: "People who build interface and build products don't go to ACM (Association for Computing Machinery conferences like they used to. And when you go to practitioner events, yo don't see professors."

In the 1990s, Carroll's work on minimalist documentation also influenced an IBM technical writer to architect the original DITA schemas for structuring content. That technical writer was Michael Priestley, who a few years later co-wrote (with JoAnn Hackos) the original technical specification for the DITA standard and is currently co-writing (with me) the spec for LwDITA. Priestley has mentioned on many occasions the impact and influence of Carroll's work on DITA and LwDITA. Therefore, Carroll is also "blamed" for the development of DITA. "John Carroll's work on minimalism was part of the context of DITA," Priestley told me.

> DITA just encoded some existing writing standards within IBM, for topic orientation and information typing. It wasn't until we went public with DITA and people started challenging those choices that I went back and tried to understand where they came from. Minimalism was one of the big influences on DITA, but I first received the influence filtered through guidelines written by others, including the editorial standards in the IBM publication Developing Quality Technical Information.
>
> (Priestley, personal communication, 2017, November 21)

Priestley remembers that, a few years ago, someone questioned on a practitioner's blog if DITA was indeed a product of minimalism applied to technical instruction. As a result, he went back to Carroll's original writing in *The Nurnberg Funnel* and found "the prototypical example, the user guide that Carroll used to test out the principles." He discovered that "you could write that user guide in DITA today. I was surprised and gratified by how close the match was, and by how resilient and persistent those principles have proved." That concept of applied minimalism is an undeniable engine behind the concept of intelligent content, which would not exist if authors were still writing print-based manuals with unstructured sections and chapters.

So, how does John Carroll plead when confronted with the claim of DITA ancestry?

> If DITA was inspired by my work, then thank you. However, that mostly happened after me (at IBM). This idea of adding metadata to facilitate modular reuse of text, I first encountered it with a (female) technical communicator at IBM, at the Raleigh lab in the early 90s. It was an interesting idea because people were talking back then about object-oriented software, modular units . . . all that kind of object-oriented thinking that was being applied to text in a pretty interesting analogy.
>
> (Carroll, personal interview, 2015, May 28)

I tried to identify the technical communicator from the IBM lab in Raleigh who introduced Carroll to DITA. Based on conversations with former and current IBM employees, I determined that the technical communicator could be

Susan Carpenter, who is now a senior manager of technical documentatio at Red Hat and "led the first pilot of DITA with IBM WebSphere Applicatio Server docs" (DITAWriter, 2016). Through an online introduction by Micha Priestley, I received the following reply from Carpenter:

> It was probably me. I was leading IBM's production pilot of DITA i 2001/2, and I definitely had contact with him (Carroll) during that tim I don't remember specifically talking about minimalism with him, but was very much on my mind. We stripped a lot of bloated material out inventory as part of our conversion to DITA.
>
> (Carpenter, personal communication, 2018, May 1

A few weeks after I interviewed John Carroll, the influential technical com munication blogger Tom Johnson talked to him at the Society for Technic Communication Summit. Johnson questioned if DITA was a product of min malism in documentation.

> I had the opportunity to ask John Carroll what he thought of DITA, sinc many DITA people claim Carroll's minimalism as foundational to tec comm. Carroll said he's always elated to see people take his work an incorporate it into their approaches. At the same time, he said any syste that traps you into a fixed pattern or template can be detrimental if th content doesn't fit that pattern or template.
>
> (Johnson, 201

Michael Priestley claims that the DITA content types of concept, task, an reference are an implementation of Carroll's idea behind the minimal manua The fixed content types that have been a part of DITA XML since its origins IBM have always been open to evolution: an author can write, for example, task topic with the default elements and attributes included in the DITA stanc ard. That same author could, with some tweaking of XML schemas or with hel from an information architect, adapt the task topic to make the "fixed patter or template" mentioned by Johnson into something that works for them. I Chapter 1, I mentioned an example of Chef Pedro, who could *specialize* el ments of the DITA task content type into more meaningful components for h intended purpose (i.e., **<ingredients>** as a specialization of the default **<prereq** for pre-requisites). This process of specialization is reflected in the "D" DITA: topics and their elements can evolve in the Darwinian sense of the wor However, that process of evolution is not necessarily transparent for autho who do not want or do not know how to edit XML schemas. By simplifyin the core "fixed pattern of template" of DITA, LwDITA embarks on a process genre evolution that empowers entry-level users while keeping its tools to sup port our human obsession with order and rules.

Our Obsession with Order and Rules

The demise of the manual is an event that affects genre theory and technical communication, as this was the flagship format that the profession produced for decades and, for some, justified its existence as an academic subject and area of practice. Nobody navigates that crossroad of genre and technical communication like Carolyn Miller. Even after retirement from her faculty position at North Carolina State University, Miller continues her explorations on the human need for genres in culture, literature, and the sciences "to help us make sense of this blooming, buzzing confusion" of new forms of communication and interaction (Miller, 2015, p. 155).

Using Darwinian terminology that resonates with design principles of the DITA and LwDITA standards, Miller noted that "in trying to understand the process of genre change and the emergence of what seem to be 'new genres' in both new and old media, we have come to rely heavily on the concept of 'evolution'" (2015, p. 155). As she traced the use of evolutionary language when describing genre use in literature, architecture, music, and painting, Miller attributed this appropriation of language from biological sciences to an "obsession with order and rules" (2015, p. 162). That same obsession is behind the methodology of structured authoring of technical information, which is all about "capturing, guiding, and validating the content, order, and form of a piece of content" (Baker, 2013, p. 189). It is also behind the predetermined content types (e.g. concept, task, and reference) in the DITA standard. The obsession shaped the old flagship genre, as heavy computer manuals established rules for data processing, ordered sections according to product features, and forced readers to follow a specific instruction and consultation pattern.

Taken to its most dangerous extreme, this obsession with order and rules can lead to the standardization of cultural products that Theodor Adorno (1991) presaged. In contrast, taken to its most beneficial extreme, obsessing about order and rules can produce information schemas as revolutionary as those proposed by J.C.R. Licklider (1965) for cataloging library sources. For technical communicators creating intelligent content, the negative side of schemas (probably presented as XML files) is in the "perception that XML forces writers into creating cookie-cutter topics rather than useful technical information" (O'Keefe, 2010, p. 37). A positive side comes from the order and consistency in content that structured authoring methodologies enable.

> Therefore, structured writing is not the enemy of professional writers, but a natural and proper part of their professional tool chest. And for occasional writers – those whose main job is something else but who are sometimes called on to produce content – structured writing can be a godsend if implemented properly. It guides authors and lets them know what is required and when they have completed the task in a satisfactory manner.
>
> (Baker, 2013, p. 189)

The process of identifying, documenting, and implementing commo structures to preserve order and rules in the content produced by technic and professional communicators follows a longstanding tradition of applie rhetoric in writing studies. For example, Linda Flower explored transferabl tasks in professional writing through the use of schemas and writing plan acknowledging that "many discourse conventions are, in fact, formalizations c rhetorical moves" (Flower, 1989, p. 34). Studies about rhetorical convention in technical and professional communication genres have not disappeared, a Kim Sydow Campbell and Jefrey S. Naidoo showed on their study of structur moves in marketing white papers (Campbell & Naidoo, 2017). For a discours community of technical communicators producing computer manuals befor the dawn of minimalism in technical communication, those rhetorical conven tions were documented and analyzed in computing-related publications – an those are the origins of LwDITA and intelligent content.

Common Structures in Technical Content Before Minimalism

The October 1975 edition of *Asterisk* (the "Systems Documentation Newsletter from the Association for Computing Machinery) included a glimpse of a techn cal manual's rhetorical conventions. In the "Technical Writing" column, edite by Diana Patterson (she of the prestigious ACM SIGDOC Diana Award), Larr Wygant, of CAPONE – Chicago Area Programmers of Novas and Eclipses, rec ommended some sections for user manuals aimed at attracting the attention c a reader, who was expected to be male in the "highly selective environment" c old computing described by John Carroll earlier in this chapter.

> I feel very strongly that when one opens a manual, the first printed pag to fall to the eye should be a very general explanation of exactly what th system is, what problem it was intended to attack, and the reasons fc bothering to attack that problem at all. This is somewhat akin to the 'hool at the opening of a novel; if the readers' attention is not captured imme diately, he is likely to wander off on any number of tangents and end n better informed than he began.
>
> (Patterson, 1975, p. 8

Wygant's analysis of a user manual then focused on the criteria behin the extent and cost of documentation: content, intent, and technique. Whe it came to content, he recommended to "(s)uit the complexity of the manu to the complexity of the subject, and don't belabor the obvious." He define intent as the goal to cover the users' needs: "The documentation should b clearly aimed at the prospective user of the system. As many levels of docu mentation are required as there are levels of users." Technique was connecte to the budget and quality of documentation: "Two elements of documentatio

are indispensable – integration with the system implementation and on-line machine-readable generation" (Patterson, 1975, p. 8).

A later issue of *Asterisk* included a more detailed "User Manual Outline," authored by Joe Rigo (he of the prestigious ACM SIGDOC Rigo Award). Rigo's outline was "designed mainly for a software vendor or equivalent system application. It is a manual of instructions on how to use the program or system. It assumes a batch, or similar non-interactive, application" (Rigo, 1976, p. 7). The sections identified in Rigo's outline provide a rich description of the genre's rhetorical conventions (or at least those identified by the author), which include the following:

Preface ("less than 1 page"). Should identify the system by name and state its purpose, intended audience ("by job title or profession") or special training prerequisites, hardware and software requirements, "(c)opyright restrictions, address for questions and comments, list of other directly relevant publications, etc." "None of these topics should extend beyond a single short paragraph. Some can be handled with a single sentence. The rest of the manual can be used to expand where necessary" (1976, p. 7).

Introduction. Should describe "the program or system as it will appear to the user. Identify its main functions and refer to an appropriate section later in the manual for detailed information." "The complete introduction should not be more than 1–3 printed pages. It is really not much more than a long-winded Topic Index" (1976, p. 7). The Introduction should also note any documentation conventions used in the manual (all caps means that words should be coded exactly as printed, lower case describes a user entry point, braces represent a mandatory choice, brackets represent optional parameters, etc.) Rigo added that if "the system does not function as described, users will quickly lose faith in the manual. On the other hand, they will accept almost any outrageous restriction if it is identified and described clearly" (1976, p. 7).

Basic requirements. Should include "any information that the user needs to determine whether he can run the system in his installation or use it for his application. This information may be moved to an appendix if it is more than 2–3 pages or is primarily technical in a manual that is intended for people with no programming experience" (1976, p. 8).

Basic functions. Should contain "step-by-step instructions for performing a task of primary interest to the user. It shows how different elements of the system are used together on an intermediate level" (1976, p. 8). This section "introduces new users to the system. Experienced users will refer directly to the reference material in other chapters. Do not assume that a person reading these sections knows anything about the system" (1976, p. 9).

Creating the data base (sic). "This section does not apply to all software packages (sic), and its contents are variable even when it does apply. It may be moved to an appendix if the process is performed only once for the entire installation" (1976, p. 9).

Processing and reporting. Should describe "all statements, command parameters, and options that a user must cope with" (1976, p. 9). In Rigo's out line, "each statement, command, option, parameter, or similar independer item" requires the name of the statement or command, a paragraph describin its function, a formal definition using the documentation conventions stated i the Introduction, examples of valid commands/statements, summary of rule and additional examples as needed. Rigo closed this section stating that th "manual must then include a reference chart showing all command formats o a single page" (1976, p. 9).

Sample output. Should contain "little more than an annotated listing fror a typical run." Rigo specified that this "section does not have to explain every thing" (1976, p. 9) and it should advise the user "that this is all he needs to kno about the subject unless he wants to become a computer programmer, in whic case he should read some other book" (1976, pp. 9–10).

Complete examples. Should contain "at least one complete example." "D not try to show off every bell and whistle in the system in a single example. Th simply creates confusion" (1976, p. 10).

Error messages. "List each error message that a user may encounter. Explai what the user must do to eliminate the error condition. Be specific. No erro message is self explanatory" (1976, p. 10).

Appendices. "An appendix is the purest, most usable form of computer sys tem documentation. It is usually well defined and restricted to a single narro topic. As a result, it becomes very easy for a user to find exactly the informatio that he is looking for" (1976, p. 10). Rigo concluded his ode to the appendix wit the following: "Appendices are great. Use them" (1976, p. 10).

Supplemental. Should include a table of contents and an index. "A page o 'How to . . . ' references is also helpful. Glossaries are rarely worth the effort. have never yet used one that provided the explanation that I needed. It is com mon to include a list of illustrations, but no one seems to use them very much (1976, p. 10).

Rigo's proposed sections and moves establish a clear pattern or schema fo composing an example of the technical manual as a genre of technical com munication. The conventions he identified had the purpose of keeping th document's contents in order and make it easier to access for a reader.

Putting the Manual on Life Support

Joe Rigo's outline for a computer manual reflects a healthy and confident genr of technical communication in the 1970s. However, the manual started to los genre power in the 1980s and was on life support even before John Carro delivered the final uppercut with the concept of minimalism. Gerald Cohen an

Donald Cunningham, in the introduction to their 1984 book *Creating Technical Manuals: A Step-by-Step Approach to Writing User-Friendly Instructions*, already talked about a "crisis in manual writing." According to these authors, manual readers in the 1980s were "more sophisticated and liberated" than those from before 1970 and would not "tolerate a poorly conceived manual." Many of their recommendations urged the manual writer "to serve as an agent representing the reader." One of the greatest failures of manual writing, they wrote, is "the lack of representation from the reader needs" (Cohen & Cunningham, 1984, p.13). They provided the following causes behind the crisis in manual writing:

1. An "increasing reluctance to read instructions, whether they are on a product, on its packaging, or in a manual." They hypothesized that conditions like "the age of television we live in" or the bad reputation spawned by manuals of the past were to blame.
2. A more discerning audience, aware that users' "inability to understand a manual is *not* a reflection on them but on those who created and published the manual.
3. A need "to get the reader productive right away," claiming that a reader "cannot, and will not, spare the time for a long period of learning for a product or procedure that is supposed to be simple and easy to use or do, maintain, or repair" (Cohen & Cunningham, 1984, p.1).

Cohen (then a technical writer at IBM) and Cunningham (then a professor of technical and professional writing at Texas Tech University) were quite prescriptive when documenting rhetorical conventions of the manual. They advocated for the use of a series of plan sheets that functioned as the manual's blueprint: "Just as a house should not be built without a blueprint, a manual should not be created without a blueprint – not just an outline, but a blueprint" (1984, p.12). Their recommended plan sheets were to be initially completed by subject-matter experts (actually, the authors said that writers should "use" subject-matter experts, with awareness that the experts "are busy people"), who would provide information "in a form that is close to the coherent pattern you will use in writing the manual. With the completed plan sheets, you will be able to determine systematically and with reasonable completeness the information you need to write the parts of the manual" (1984, p.14). The plan sheets from Cohen and Cunningham covered the following conventions of the manual as a genre of technical writing:

- Naming the Product or Procedure
- Explaining What the Product or Procedure Does
- Translating Technical Facts

- Explaining the Distinguishing Characteristics of the Product or Procedur
- Illustrating the Old Product or Procedure
- Illustrating the New Product or Procedure
- Outlining the Task
- Detailing the Task
- Presenting the Alternatives Sheet
- Presenting the Troubleshooting Table (Cohen & Cunningham, 1984).

Although the recommendations from Cohen and Cunningham were focuse on a manual that was to be presented as a book, they predicted an era "whe advanced technology will enable us to use on-line documentation exclusively (1984, p.146). They presented the possibility of online documentation as "a toc radically different from the traditional user's manual" that would require a ne approach to composing prose.

Focusing on a different genre of technical writing, Carolyn Miller and Jac Selzer analyzed the structure and argumentation elements of technical report by detailing their rhetorical conventions, or topical basis of discourse. The proposed three kinds of topics: "Those specific to a genre, those specific to a organization or institution, and those specific to a discipline" (Miller & Selze 1985, p. 311). Following their model, the rhetorical conventions of a technic manual presented by Cohen and Cunningham can be described as generic they were specific to a genre (the technical manual). The authors did not preser them as, say, the official rhetorical conventions for manuals at IBM. Jonatha Price, however, did move beyond the generic conventions and crossed into th organizational or institutional type of topical convention in his 1984 book *Ho to Write a Computer Manual: A Handbook of Software Documentation*. Pric was a senior technical writer at Apple, and described the book as related t organizational practices,

> This book began at Apple Computer as a guide to the new employees an freelance writers working for the User Education group in the divisio that produces the Apple II family of computers.
>
> (Price, 1984, p. v

Price's conventions and recommendations were presented with the ultimat goal of creating "friendly manuals," which the author posited as the antidot against bad manuals of the past. "Friendly manuals sell products, expan users' understanding of the extra features, and save everyone's time," he wrot (Price, 1984, p.7). In a meta application of his own recommendations, Price description of the features in *How to Write a Computer Manual: A Handboo of Software Documentation* reflected the organizational patterns of the type c friendly manual promoted in the book.

1. A table of contents, which lists all the main sections at the beginning of the book. Also, the start of each chapter includes a table of contents.
2. Lots of headlines, which allow the reader to "skip to the part of the book you want."
3. Checklists of key points, which are presented at the end of every chapter and then collected at the end of the book.
4. A glossary, which defines terms mentioned in the book and is presented at the end of the book.
5. An index, which is organized alphabetically and also presented at the end of the book (Price, 1984).

Some of Price's recommendations transcended to the third kind of rhetorical convention proposed by Miller and Selzer, as they became disciplinary moves or commonplaces. Price's separation of a manual's text into content types establishes a distinction between tutorial ("offers step-by-step training focused on a particular activity") and reference ("offers procedures that users can apply in many different circumstances") writing conventions. Those moves advanced from the organizational realm (Apple Computer, in his case) and became disciplinary conventions: task and reference (along with concept) are the main content types associated with the DITA standard.

Price also included a whole chapter dedicated to "Creating Computer-Assisted Instruction." When he compared online documentation to its printed version, he claimed that "people find them friendlier (At least it can be friendlier. That's up to you)" (Price, 1984, p.100). He warned that creating online documentation generated new demand for skills in the technical writer's repertoire, including learning an authoring language or, "if you don't find an authoring language you enjoy," collaborating with a programmer. Price's discussion of online deliverables advanced the conversation on topic-based authoring that moved away from the traditional manual-as-book model that dominated the market in the 1970s and before. This move also planted the seed for disciplinary conventions, as topic-based authoring for online deliverable replaced the long print format of documentation.

Price's emphasis on task-orientation established some topical moves that still apply to tutorial/procedural content, with the overall recommendation of focusing on organization. Those moves included the following:

- Introduce each section
- Divide your material into short steps
- Show people how to get out
- Separate what to do from what it means
- Put in lots of displays
- Define your terms

- Put in pictures
- Anticipate variations
- Summarize
- Give people a break
- Allay anxiety
- Test it. Revise it. Test it again
- Tell people where to go next (Price, 1984).

Many of those moves are captured in the element types of the DITA 1. standard (e.g., a task is "what to do" and a concept is "what it means"). Th element types in DITA also inherited generic, institutional, and disciplinar conventions from another title that was originally published in the 1980s: IBM *Producing Quality Technical Information.*

Developing Quality Technical Information

When I tried to measure the impact of John Carroll's theory of minimalisı on content development, I remembered that Michael Priestley has acknow edged in several occasions that minimalism, via the IBM publication *Developin Quality Technical Information* (*DQTI*), was one of the big influences on DITA In a conversation we had after one of our presentations at a DITA Nort America conference organized by the Center for Information-Developmer Management, Priestley pointed out that IBM's *DQTI* was "all about DITA with out mentioning DITA by its name."

Before *DQTI*, however, there was *PQTI.* The publication's original title wa *Producing Quality Technical Information*, and it started as an internal IBM handbook of recommendations for content authors and evaluators. Publishe in 1983, *PQTI* was credited to Morris Dean and a long list of co-authors (or cc "preparers") and focused on a list of requirements of quality, presented unde the principle of "technical information that meets all the requirements is qualit information" (Dean et al., 1983, p. ii). The structure of *PQTI* is built around list of quality requirements, and a standard pattern of presenting two example when discussing each requirement: one showing a common error and the sec ond illustrating a way to correct the error and "improve quality."

The original quality requirements from *PQTI*, organized in categories an specific actions under each category, were the following:

- Task orientation
    - Present information from the reader's point of view
    - Indicate a practical reason for information
    - Order the presentation to reflect the order of use
    - Devise titles and headings to reveal the task

- Organization
  - Reveal how the pieces fit together
  - Emphasize main points; subordinate secondary points
  - Don't force readers to branch unnecessarily
  - Present similar topics in a similar way
- Entry points
  - In introductory sections, reveal the order of topics to follow
  - Stock the index with predictable entries for the topics covered
  - Highlight key terms – including new terms being defined
  - Rarely run text for half a page without a heading
  - Rarely run a paragraph beyond a dozen lines
- Clarity
  - Present material so readers can understand it the first time
  - Pace the presentation to be neither too fast nor too slow
  - Write directly and economically
  - Use only technical terms that are necessary and appropriate
  - Define each term new to the intended reader
  - Provide appropriate examples to communicate effectively
- Visual communication
  - Attract and motivate your readers with graphic techniques
  - Employ visual techniques to communicate effectively
- Accuracy
  - Provide accurate technical information
  - Provide accurate references and other auxiliary information
  - Use correct grammar, spelling, and punctuation
  - Use sexually neutral terms, unless other terms are appropriate
- Completeness
  - Cover all the topics that readers need, and only those topics
  - Cover each topic in just as much detail as readers need
  - Include all standard parts and all promised information
  - Repeat information only when readers will benefit from it (Dean et al., 1983).

Some of Dean et al.'s recommendations reflected major changes for documentation from that period. These moves included advocating for task-oriented titles and headings in sections and the use of the term "topic" (years before the development of a DITA topic model) as a component in a "library" of information units that a writer used to assemble documentation deliverables. Other

recommendations, however, were standard guidelines for technical writer Table 1 shows a list of "weak verbs" and "roundabout expression" compared t "strong, precise verbs" and "concise terms" from the "Write directly and ecc nomically" recommendation under the requirement of "Clarity."

The combination of big-picture requirements and concrete recommenda tions responds to *PQTI*'s dual audience. Early in the document, the autho specified that "this book is to help writers produce quality technical informatio and to help reviewers judge the results" (Dean et al., 1983, p. ii). Therefore, ider tifying genre conventions in this text was not as direct as it was in other how-t titles about creating documentation.

In 1998, *PQTI* evolved into *DQTI* – the first edition of *Developing Qualit Technical Information: A Handbook for Writers and Editors*, authored b Gretchen Hargis and published by IBM/Prentice Hall. Just as the origina publication, *DQTI* focused more on recommendations than in specific topi cal conventions. However, in *DQTI* the quality requirements became "qualit characteristics," presented as a "system for editing and evaluating technica information" (Hargis, 1998, p. xiii). Additionally, Hargis organized the charac teristics "by their ability to make information" in the following groups: easy t use, easy to understand, and easy to find (Hargis, 1998, p. 2).

Hargis specified that the quality characteristics were not presented as generi conventions or, as she called them, "elements". She defined elements as "th units that physically make up technical information. An element can be as sma as a word and as large as a tutorial" (1998, p. 3). She presented headings, list and tables as examples of those elements, indicating that whereas in works c

TABLE 2.1 Excerpt from the "Clarity" requirement of *Producing Technical Informatio* which compares "weak verbs" to "strong, precise verbs" and "roundabout expressions" to "concise terms" (Dean et al., 1983, p. 24)

Weak Verbs	*Strong, Precise Verbs*
• has the capability	• can
• is capable	• requires
• has a requirement	• agrees
• is in agreement	• prints
• performs the printing	• infers
• draws a conclusion	• helps
• provides assistance	
Roundabout Expressions	*Concise Terms*
• at this (that) point in time	• now (then)
• due to the fact that	• because
• in the event that	• if

fiction a paragraph might be the main element, in technical information authors required "other means of differentiating information" (p. 3). The guidelines presented in *DQTI*, Hargis claimed, were "brief directives about what to do or not to achieve the characteristics of quality. These guidelines apply to the elements." Further describing the difference between characteristics and elements, Hargis noted that "writers cannot, for example, turn clarity on and off with a markup tag. Elements, however, do have recognizable limits and often have corresponding markup tags" (1998, p. 3).

Hargis also mentioned minimalism as a principle for writing technical information. In her discussion of the characteristic of completeness, she included the following paragraph:

> Writers have often created technical information that is too complete, including everything there is to know about a product. Today, the trend in technical writing is toward providing the user with far less information, an approach called minimalism. The minimalist approach not only yields a reduction in page count, but it also results in writing that doesn't get in the user's way. With a minimum of information, the user can independently explore a product after learning some basic concepts and tasks. (Hargis, 1998, p.49)

A second edition of *DQTI* came out in 2004, and it added a group of co-authors after Hargis's name. The second edition provided new details on the difference between the guidelines and the elements (topical conventions) that put those guidelines into practice.

- Guidelines are "brief directives about what to do or not to do to achieve the characteristics of quality. These guidelines apply to the elements."
- Elements are "the units that physically constitute technical information. An element can be as small as a word and as large as a tutorial" (Hargis et al., 2004, p. 5).

Gretchen Hargis passed away in 2005, and the third edition of *DQTI* was published in 2014, with some of the second edition's co-authors. According to its introduction, the 3rd edition of *DQTI* focused on,

- Greater emphasis on the embedded assistance in user interfaces
- The need to plan for information access from mobile devices
- The pervasiveness of Google and other search engines as users' preferred method for looking for information
- Video as a delivery medium for technical information (Carey et al., 2014, p. xviii).

The quality characteristics in all the available editions of *DQTI* follow th same structure. The definitions for some characteristics, however, have change slightly over time. For example, task orientation was originally defined as: "A focus on helping users complete the tasks associated with a product in relatio to their jobs," (Hargis, 1998, p. 2) and in the most recent edition is presented a "In the context of a product, a focus on helping users do tasks that support thei goals" (Carey et al., 2014, p. 14). The *DQTI* quality characteristics, which sti show some resemblance to the requirements from *PQTI*, are,

Easy to use

- Task orientation
- Accuracy
- Completeness.

Easy to understand

- Clarity
- Concreteness
- Style.

Easy to find

- Organization
- Retrievability
- Visual effectiveness (Carey et al., 2014, p. 14).

Adding to the conversation about turning characteristics into elements c conventions, Carey et al. wrote the following on the 3rd edition of *DQTI*:

> You can apply the quality characteristics whether you're writing a book, page, a paragraph, a sentence, or a single word in an interface. The qualit technical information model of nine characteristics is flexible enough t support you as you develop ever smaller chunks of information to addres the changing needs of users.
>
> (Carey et al., 2014, p. 14

If the earlier versions of *Developing Quality Technical Information* wer as Michael Priestley said, "all about DITA without mentioning DITA by it name," that changed in the 3rd edition. In their analysis of the characteris tic of "Style," under the category of "Easy to understand," the co-authors c *DQTI* asked readers to "use consistent markup tagging." A few of their specifi recommendations for tagging elements focused on the generic, organiza tional, and disciplinary implementation of the characteristics into element DITA XML.

Implementation of Elements: DITA is Born

The quality guidelines for technical information that had been driving the work of authors, editors, and reviewers at IBM since the 1980s were put into practice in the content types and elements included in DITA XML. In Chapter 1, I gave a brief historical overview of the early DITA days at IBM, when the Customer and Service Information (C&SI), with work primarily conducted by Dave Schell, Don Day, and Michael Priestley started exploring XML as a language to create tags that would create templates for the genre conventions of commonly used content structures in the company's documentation. Going back to the Miller and Selzer taxonomy of topical conventions, that early work captured the generic structures of fixed types that included the classical DITA trio of concept, task, and reference. As those DITA fixed templates for topics and their structures at the section, paragraph, sentence, or even word level became widely used within IBM, the structures moved to the organizational or institutional category. In conference presentations and publications (Priestley et al., 2001; Priestley, 2001), members of that pioneer team were introducing DITA as the IBM way of authoring and publishing technical documentation. When other companies heard about DITA and tried to create their own architectures for authoring and publishing, the topical conventions moved to the disciplinary category, which started when IBM announced, on March 15, 2001 that they "were going public with an architecture for technical documentation using XML" (Priestley, 2001a), and then culminated with the release of DITA 1.0 as an open standard in 2005.

The transformation of *DQTI* guidelines into DITA elements was subtly documented in a 2001 article published in the "Technology Review" column of the journal *Technical Communication*. "DITA: An XML-Based Technical Documentation Authoring and Publishing Architecture" was authored by Michael Priestley, Gretchen Hargis, and Susan Carpenter, who have appeared previously as key players in this chapter.

The article started by acknowledging that XML "has gained popularity in the technical writing profession by offering us a logical and fairly straightforward framework for developing structured information" (Priestley et al., 2001, p. 352). Then, it warned readers about the pitfalls of implementing a custom XML solution or adopting a standard: "In other words, when you create a new markup language (using XML to define its markup and rules), you shut yourself off from interchange with the rest of the world; when you adopt a standard markup language, you lose the benefits promised by content-specific markup" (2001, p. 354).

Their proposed solution was, as we know by now, the Darwin Information Typing Architecture. The article emphasized that DITA was more than just an XML DTD (Document Type Definition) for structuring documents.

As presented in that introductory paper, a DITA-based solution for developin structured information consisted of the following three parts:

> Fix the content – Authors need to rethink how to write content to thoı oughly separate form from content.
>
> Fix the design – Architects need to rethink how to classify and desig information, to reduce the cost of upfront analysis and ongoing mainte nance for content-specific markup.
>
> Fix the process – Programmers need to rethink how to create transform and processes, to allow content-specific information to be exchanged, an to make it easier to create and maintain specialized processes.
>
> (Priestley et al. 2001, p. 354

The implementation of quality characteristics into actual XML elements an conventions is all over those three parts, but it is mainly represented in the wor to be performed by authors as part of their work to "fix the content" by categc rizing topics by type of information. Priestley and his co-authors addressed th evolution of the manual as it became a collection of topics with "fine-grained moves to signal or tag specific conventions:

> When writing software manuals, technical writers have historically distir guished task-based instructions (guidance information) from referenc information. The content and organization of a user's guide, for example, different from the content and organization of a reference manual. By wri ing topics, however, you can make finer-grained distinctions about the type c information that a user can expect than just at the level of the whole documen
>
> (Priestley et al. 2001, p. 355

In the specific case of the task topic type, some of those moves included rationale for the task ("why or when a user would want to perform this task" prerequisites for the task ("what a user should do before performing this task" responses to the actions ("what the user should see as a result of doing the action" examples ("examples of what information to enter or what to do"), and post requisites for the task ("what to do next after this task is completed") (Priestle et al. p. 356). This highly structured task topic reads like a smoothie that cam out of a blender combining generic, organizational, and disciplinary conver tions and commonplaces from Carroll, Price, and Hargis and her co-author and others who attempted to capture the essence of technical documentation.

To illustrate some of those moves, which were present in the early DITA tas topic and are still part of the standard, consider the marinara sauce recipe fror Chapter 1 (Figure 2.1). I added some extra elements (in bold font) to implemer the information types mentioned by Priestley et al. in the previous paragraph.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE task PUBLIC "-//OASIS//DTD DITA Task//EN" "task.dtd">
<task id="t-marinara">
    <title>Marinara Sauce</title>
    <shortdesc>Prepare a crowd-
pleasing red sauce for pasta in about 30 minutes.</shortdesc>
    <prolog>
        <author>Unknown</author>
        <metadata>
            <category>Italian</category>
        </metadata>
    </prolog>
    <taskbody>
        <prereq>
            <ul>
                <li>2 tbsp. of olive oil</li>
                <li>2 cloves of garlic, minced</li>
                <li>1/2 tsp. of hot red pepper</li>
                <li>28 oz. of canned tomatoes, preferably San Marzano</li>
                <li>2 tbsp. of parsley, chopped</li>
            </ul>
        </prereq>
      <context>Prepare this sauce in the summer for an Italian-inspired
afternoon</context>
        <steps>
            <step>
                <cmd>Heat olive oil in a large saucepan on medium</cmd>
            </step>
            <step>
                <cmd>Add garlic and hot red pepper and sweat until fragrant
</cmd>
            </step>
            <step>
                <cmd>Add tomatoes, breaking up into smaller pieces</cmd>
            </step>
            <step>
                <cmd>Simmer on medium-low heat for at least 20 minutes</cmd>
            </step>
            <step>
                <cmd>Add parsley</cmd>
      <stepxmp>Sprinke the parsley all over the saucepan</stepxmp>
       </step>
  <step>
         <cmd>Simmer for another five minutes</cmd>
      <stepresult>The Marinara should be thicker than regular tomato
sauce</stepresult>
                   </step>
            <step>
                <cmd>Serve over long pasta.</cmd>
            </step>
        </steps>
      <postreq>Don't forget to clean the kitchen after dinner</postreq>
</taskbody>
</task>
```

FIGURE 2.1 Expanded version of the marinara sauce recipe as a DITA task. Elements in bold type emphasize some of the moves of interacting with users that the DITA creators had in mind when developing the document types.

The rationale for the task is in the **<context>** element; it represents th conventional move of telling the user when or why to complete the procedur recipe. The prerequisites were already present in the example from Chapter in the **<prereq>** element, which includes an unordered list (**<ul>**) with list item (**<li>**) for all the ingredients the user would need to complete the recipe. Th element **<stepresult>**, which is a child of the element **<step>**, provides possibl responses to the actions conducted by users. Another child of the **<step>** el ment, **<stepxmp>**, gives an example of how to conduct a specific command c action. Lastly, the **<postreq>** element can provide one or more actions that nee to be conducted after the core steps of this specific task topic.

Those conventions, or content commonplaces, specific to the genre of tas identified by the DITA architects, created quite the strict template that cam with many of the benefits I have mentioned in previous chapters. One of th many approaches I have adopted to introduce the DITA standard in a humar ities-based Professional and Technical Writing curriculum is to present thos elements as rhetorical conventions of our specialized discourse communit This approach also helped during the process of designing the content an element types of LwDITA, but it all started with understanding the differer meanings of the term *topic* for the proposed standard's intended audiences c practitioners and academics.

A Topic by Any Other Name . . .

In the world of DITA XML, "a topic is the basic unit of authoring and reuse (2.2.1.1 The topic as the basic unit of information, 2016). The technical specif cation for the DITA standard defines topic as follows:

> DITA topics consist of content units that can be as generic as sets of para graphs and unordered lists or as specific as sets of instructional steps in procedure or cautions to be considered before a procedure is performe Content units in DITA are expressed using XML elements and can b conditionally processed using metadata attributes.
>
> (2.2.1.1 The topic as the basic unit of information, 201

In the world of rhetoric that defines discourse in most academic program teaching technical communication in the United States, a topic would actu ally be one of those content units, and not necessarily the whole "DITA topic. Confusing? Imagine being a student in Professional and Technical Writin and reading the books *Topic-Driven Environmental Rhetoric* and *DITA – th Topic-Based XML Standard* for different courses on the same semester. The *top ics* from the first book are not the same *topics* in the second book. Or are they?

The origins of DITA introduced the topic as a chunk of information with th following characteristics:

- One subject, signified by the title
- Wording that is independent of any other topic
- Appropriate length to treat the topic adequately yet not require lots of scrolling (Priestley et al., 2001, p. 355).

Trying to define the concept of topic in rhetoric is not that easy. Derek Ross warns readers about the deceptive simplicity of the terminology for topics in Aristotelian-inspired discourse:

> Aristotle's lack of definition is perhaps not surprising given the seemingly simple concept of "topic." Even today we think we readily understand what is meant by a "topic": We know that the "topic" of a conversation, for example, is that thing upon which the conversation hinges. Similarly, to some extent, we understand the idea of a commonplace without much prompting: A commonplace idea is one which is, quite literally, common to a particular place. The perception is that little explanation is needed.
>
> (Ross, 2017, p.6)

Ross summarizes the discussion by suggesting that topics are shared places common to all debate, which allow rhetors to shape viable argument, and that is where the rhetorical concept of topic meets the DITA topic. The rhetorical topics are the moves/elements inside a DITA topic: the element of, say, **<prereq>** for prerequisites in a task topic suggests a shared model common to all tasks and procedures that allows authors to produce viable how-to information. A definition of topic from Lawrence Prelli further supports this overlap. Prelli explains rhetorical *topoi* as "headings or topics that identify lines of thought." He adds that "some of these lines of development are relevant to virtually all discussion within a culture; others are peculiar to specific subjects and fields of inquiry" (Prelli, 1989, p. 185). Some DITA XML elements, like **<p>** for paragraph or **<li>** for list item, are relevant for any type of information and are present in HTML – the *lingua franca* of the web. Other elements, like **<lcObjective>** for a single learning objective or **<lcDelivery>** for delivery method of educational content, are peculiar to the DITA learning and training specialization.

William Hart-Davidson was an early DITA adopter in academia. His WRA 420: *Advanced Technical Writing* course at Michigan State University had students using the DITA standard to structure and publish information more than a decade before I started writing this chapter. Some of Hart-Davidson's work has focused on that overlap of the topic as content unit and the topic as commonplace element (Hart-Davidson & Omizo, 2017). Particularly, the course content he developed for the ELI Review modules on *The Essential Moves of Technical Communication* (Eli Review, n.d.) is based on common traits of technical communication genres. In a conversation we had a few years ago when he

TABLE 2.2 Attempt to map the essential moves of technical communication from the ELI Review curriculum to content elements from the DITA standard

Essential move of technical communication	*DITA element*
Staging: explaining the goal to be achieved	**<shortdesc>**
Staging: explaining conditions necessary to begin	**<context>**
	<prereq>
Coaching: describing steps in a process	**<steps>**
Coaching: describing success conditions	**<stepresult>**
Alerting: helping users avoid unwanted outcomes	**<info>**
	<note>
	<hazardstatement

visited the Virginia Tech campus to give a talk precisely on rhetorical topics an commonplaces, Hart-Davidson linked those traits or moves to elements fro the DITA standard, which I tried to capture in Table 2.

The overlap, despite apparent and actual differences, in the DITA XM and rhetorical discussion of topics and commonplaces highlights the practic applications of topics, and also their perceived limitations. Miller and Selze analyzed rhetorical topics as repetitive but useful patterns of speech in the fo lowing paragraph:

> Topics can thus be conceived, alternatively, as pigeonholes for locatin already existing ideas or as patterns of thought or methods of analys that can be called on in the construction of arguments. In practice, thes alternative versions are conjoined: by learning a mechanical system pigeonholes, one masters patterns of thought that then become habitu and spontaneous.
>
> (Miller & Selzer, 1985, p. 31

In DITA XML, that "system of pigeonholes" (the standard's elements an content types) allows authors to master specific forms of communication tha enable the type of intelligent content discussed in this book. The next sectio focuses on the number of elements included in the DITA standard across i releases. It includes some examples showing the useful potential of those ta while setting the exigence for LwDITA's simplified content model.

An Element for Every (Content) Occasion

One could point out that the number of XML tags available in the DITA stand ard is the result of years of committee-based development with active membe representing different sectors and industries with unique content-structurin

needs. However, DITA was born a heavy spec. Even in its version 1.0, the standard had a total of 193 element types. Some of those original element types are now deprecated, but the specification for DITA 1.0, edited by Michael Priestley and JoAnn Hackos, included the following elements and categories:

- Topic elements: 11, including **<dita>, <topic>**, and **<title>**
- Concept elements: 2, consisting of **<concept>** and **<conbody>**
- Reference elements: 12, including **<reference>**, **<refbody>**, and **<refsyn>**
- Task elements: 25, including **<task>**, **<taskbody>**, and **<steps>**
- Body elements: 29, including **<ol>**, **<p>,** and **<ul>**
- Table elements: 11, including **<table>**, **<row>**, and **<entry>**
- Typographic elements: 6, including **<b>**,**<i>**, and **<u>**
- Programming elements: 25, including **<codeblock>**, **<apiname>**, and **<syntaxdiagram>**
- Software elements: 8, including **<varname>**, **<userinput>**, and **<systemoutput>**
- User interface elements: 5, including **<wintitle>**, **<menucascade>**, and **<screen>**
- Utilities elements: 4, including **<imagemap>**, **<area>**, and **<shape>**
- Miscellaneous elements: 5, including **<draft-comment>**, **<fn>**, and **<indexterm>**
- Prolog elements: 27, including **<audience>**, **<author>**, and **<platform>**
- Related links elements: 5, including **<link>**, **<linkinfo>**, and **<linktext>**
- Specialization elements: 6, including **<boolean>**, **<no-topic-nesting>**, and **<required-cleanup>**
- Map elements: 10, including **<map>**, **<navref>**, and **<topicref>**
- Map group elements: 2, consisting of **<topicgroup>** and **<topichead>**

The original DITA 1.0 technical specification is still available on the DITA Technical Committee website[1]. The spec is a well-organized document that presents details and examples of common topic and element types without overwhelming the readers with a force-feed of its 193 XML tags.

For version 1.3 of the DITA spec, the Technical Committee actually separated the standard's 621 available elements in three editions: base (with 189 elements), technical content (including the elements from the base edition and 251 additional elements), and all inclusive (including the elements from base, technical content, and 181 additional elements). This separation responds to the expanded audiences of the standard. DITA 1.0 was faithful to the IBM original and was intended mainly as a grammar to structure software documentation. DITA 1.3, in contrast, has users in diverse industries that share a need for structured information and versatile publishing, but do not have a clear need for concept, task, and reference as core content types.

The DITA Technical Committee at OASIS released a white paper/committee note titled DITA 1.3: Why Three Editions? that emphasizes "that topic and

map are the base document types in the architecture" (Eberlein et al., 201 p. 7). Therefore, those are the only content types present in the base editio The technical content edition adds the technical communication content typ (concept, task, reference) that defined DITA in its origins at IBM, and ad the content types of glossary entry, glossary group, and troubleshooting. Th all-inclusive edition incorporates content types from the DITA learning an training specialization (learning assessment, learning base, learning conten learning overview, learning plan, and learning summary). The DITA Technic Committee describes the three editions as follows:

- "The base edition is designed for application developers and people wh need only the most fundamental pieces of the DITA framework"
- "The technical content edition is designed for authors who document com plex applications and devices, such as software, hardware, medical device machinery, and more"
- "The all-inclusive edition is designed for authors and publishers who develo and deliver well-structured, modular instructional materials. It provides framework for using a learning objects approach to organize and sequenc content as a learning deliverable" (Eberlein et al., 2015, pp. 10–17).

Providing a detailed explanation or tutorial of the content and element typ included in DITA 1.3 is way beyond the scope of Creating Intelligent Conte with Lightweight DITA. Some helpful titles that thoroughly describe an explain the standard have already been published (Hackos, 2011; Bellamy et al 2012). Additionally, the website Learning DITA[2] from Scriptorium is a free an solid resource for novice-to-advanced users who need to learn about DITA an its content structuring and publishing capabilities. However, to put the thre editions in context I include here some examples of representative topic type Going back to the story of Chef Pedro from Chapter 1, a topic type from th base edition would look as Figure 2.2, which includes biographical informatio on Chef Pedro. The topic works as a simple template that separates conte from presentation and leaves it ready to join a topic collection and be integrate into information products.

The topic's first line is the XML declaration that, as I tell my student announces to the computing processors that the topic is an XML file and shoul be treated as such. The second line points out that the topic is not just a regul XML file, but that is complies with the structuring rules described in the docu ment type definition (DTD) of "topic" from the DITA standard. This means th if an author decides to include an XML tag that does not belong in the DIT topic DTD, the file will not validate and will report an error when processe The third line opens the actual topic (with the required attribute for a uniqu identifier value: "pedro-bio" in this case). And then the file has a few move genre conventions of the introductory topic: a short description (**<shortdesc>**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "topic.dtd">
      <topic id="about">
       <title>About Chef Pedro</title>
       <shortdesc>Executive Chef, Yucatec-International Fusion Alchemist,
Marketing Architect.</shortdesc>
        <body>
      <p>Self-taught and persistent in the kitchen and marketing aspects of
the restaurant business, Chef Pedro has established himself a key player in
the new scene of Mexican gastronomy, particularly in the tradition of dishes
and flavors from the Yucatan peninsula.</p>
      <p>From an early age, Pedro learned the craft in the always-open
kitchen of his mother, Doña Raquel, among aromas of truly mestizo gastronomy.
Original top notes of condiments and herbs mixed with traditional Spanish
middle notes of saffron and olive oil built on ancient Yucatec base notes of
roasted tomato and hot peppers salsa inspired him to experiment with his own
style of cooking.</p>
       </body>
      </topic>
```

FIGURE 2.2 DITA topic including biographical information about Chef Pedro. The topic's elements include simple structures like a title, a short description, and a body containing paragraphs.

the topic's body (**<body>**), and some paragraphs (**<p>**). When placed with other topics in a collection and processed by a DITA-aware software tool (I will talk more about tools in Chapter 5), this biographic introduction can inherit any formatting rules and become a page or section in a print magazine or book, a page or blurb on a website, and many other end-user deliverables depending on the author's publishing goals.

For the technical content edition, a task topic can provide an example. Earlier in this chapter, Figure 2.1 included a modified version of the marinara sauce recipe from Chapter 1. Similar to the generic topic from Figure 2.2, the task's first line of code announces to the computing processors that the topic is an XML file and should be treated as such. The second line points out that the topic is not just a regular XML file, but that is complies with the structuring rules described in the document type definition (DTD) of "task" from the DITA standard. Then, some of the topical conventions included in the XML tags are equivalent to the essential moves of technical communication from Table 2, including **<shortdesc>** as a staging move and **<step>** as a coaching move.

The DITA Technical Committee explained that separating the DITA 1.3 spec in three editions sets "the stage for future developments: Lightweight DITA and DITA 2.0" (Eberlein et al., 2015, p. 9). LwDITA, thus, can be a truly minimalist version of the standard, giving authors the structuring and publishing capabilities of DITA 1.3 in a reduced set of tags. LwDITA also makes DITA publishing workflows more inclusive and accessible, as it accommodates non-XML authoring formats to address the needs of authors outside of the traditional technical documentation audience of DITA 1.0. Those are strong selling points that drive the next chapter.

Notes

1 https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita#technical
2 http://www.learningdita.com

References

2.2.1.1 The topic as the basic unit of information. (2016, October 25). Retrieved fro
http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/archSpec/base/topicdefine
html#topicdefined

Adorno, T.W. (1991). *The culture industry: Selected essays on mass culture*. Londo
Routledge.

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

Campbell, K.S. & Naidoo, J.S. (2017). Rhetorical move structure in high-tech marketi
white papers. *Journal of Business and Technical Communication*. 31(1) 94–118.

Carey, M. et al. (2014). *Developing quality technical information: A handbook for write
and editors*. 3rd Ed. Upper Saddle River, NJ: IBM Press.

Carroll, J.M. (1990). *The Nurnberg funnel: Designing minimalist instruction for practic
computer skill*. Cambridge, MA: M.I.T. Press.

Closs, S. (2016). *DITA – the topic-based XML standard: A quick start*. Switzerlan
Springer.

Cohen, G. & Cunningham, D.H. (1984). *Creating technical manuals: A step-by-st
approach to writing user-friendly instructions*. New York, NY: McGraw-Hill.

DITAWriter. (2016, July 13). Don Day and Michael Priestley on the beginnings of DIT
Part 1. Retrieved from http://www.ditawriter.com/don-day-and-michael-priestle
on-the-beginnings-of-dita-part-1/

Eberlein, K.J., Harrison, N., Hunt. J., & Swope, A. (2015). *DITA 1.3: Why three edition
OASIS.

Eli Review. (n.d.). The Essential moves of technical communication. Retrieved fro
http://elireview.com/content/curriculum/techcom/

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

Hargis, G. (1998). *Developing quality technical information: A handbook for writers an
editors*. Upper Saddle River, NJ: Prentice Hall.

Hargis, G. et al. (2004). *Developing quality technical information: A handbook for write
and editors. 2nd Ed*. Upper Saddle River, NJ: Prentice Hall.

Hart-Davidson, W. & Omizo, R. (2017). Genre signals in textual topologies. In L. Wals
& C. Boyle (Eds.), *Topologies as techniques for a post-critical rhetoric* (pp. 99–123
New York, NY: Palgrave Macmillan.

Johnson. T. (2015, June 29). Slides, notes, and lessons learned at the STC summ
2015 in Columbus, Ohio. [Blog post]. Retrieved from http://idratherbewritin
com/2015/06/29/lessons-learned-at-the-stc-summit-2015/

Licklider, J.C.R. (1965). *Libraries of the future*. Cambridge, MA: M.I.T. Press.

Miller, C. R. (2015). Genre change and evolution. In N. Artemeva & A. Freedman (Eds.
Genre studies around the globe: Beyond the three traditions (pp. 154–185). Lexingto
KY: Trafford.

Miller, C.R. & Selzer, J. (1985). Special topics of argument in engineering reports. I
L. Odell & D. Goswami (Eds.), *Writing in non-academic settings*. New York: Guilfor

Dean, M. et al. (1983). *Producing quality technical information*. San Jose, CA: IBM.

O'Keefe. S. (2010). XML: The death of creativity in technical writing? *Intercom*, (February), 36–37.

Price, J. (1984). *How to write a computer manual: A handbook of software documentation*. Menlo Park, CA: Benjamin Cummings.

Ross, D. (Ed.). (2017). *Topic-driven environmental rhetoric*. New York: Routledge.

Svenvold, M. (2015, January 26). The disappearance of the instruction manual. *Popular Science*. Retrieved from https://www.popsci.com/instructions-not-included

Patterson, D. (1975). Technical writing: User manuals. *Asterisk*, 2(5), 8–9.

Prelli, L. (1989). *A rhetoric of science: Inventing scientific discourse*. Columbia, SC: University of South Carolina Press.

Priestley, M. (2001). DITA XML: A reuse by reference architecture for technical documentation. *Proceedings of the 19th annual international conference on computer documentation*, 152–156.

Priestley, M. (2001a). XML and the Darwin Information Typing Architecture (DITA). *Communication Design Quarterly Review*, 2(2), 3–4.

Priestley, M., Hargis, G., & Carpenter, S. (2001). DITA: An XML-based technical documentation authoring and publishing architecture. *Technical Communication*, 48(3), 352–367.

Rigo, J. (1976). User manual outline. *Asterisk*, 2(8), 7–10.