BEST practices

A Publication of
The Center for Information-Development Management

JUNE 2022
VOLUME 24, ISSUE 2

Time to Upgrade: Making Websites Accessible to All

Sowmya Jayakirthi and Usha Rani Gunapu, VMware

In today's world, our requirement around developing technologies and modes of interaction that are inclusive and that serve greater good is becoming more obvious. Day in and day out we interact with multiple websites. While browsing, you might have come across websites that are designed well and easy to access, and a few that are not designed well. Do you know how many of these websites are accessible to the larger audience? The answer is, *not many*.

The main concept of web accessibility is that websites should be developed and designed to enable all kinds of users to access and experience the sites equally, irrespective of their physical abilities. As per the World Health Organization (WHO), there are more than 1 billion people who have some form of disability, which is around 15% of the world's population. The type of disability includes visual, hearing, motor, and cognitive impairments. Web accessibility is just not about providing a valuable experience to users with disabilities, but also to users who love to multitask, who have age-related dexterity issues, or who are in certain circumstances like slow internet, low lights, bright lights, or temporary disabilities like a hand fracture.

The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect.
— *Tim Berners-Lee*

As Technical Writers, it is our responsibility to ensure the content that we create is not only technically accurate but also accessible by all kinds of users. The best way to ensure our content is accessible is by following the Web Content Accessibility Guidelines (WCAG). Section 508 is a law that mandates websites to be safe and accessible for people with disabilities. Many countries have laws built around WCAG principles. For more information on your country specific policy on accessibility, please check Country Specific Accessibility Policy.

WCAG works on the principle that content and websites should be Perceivable, Operable, Understandable, and Robust (POUR). The guidelines are categorized around three types of conformance levels:

- **Level A** – This is minimal compliance and is easy to meet.
- **Level AA** – This is an acceptable compliance. Most of the accessibility regulations are built around this level. WCAG recommends your website should meet AA level to help cater to many users.
- **Level AAA** – This is the optimal level. This helps cater to most of the users.

Basic Guidelines

Here are a few WCAG guidelines that can be easily implemented on your websites and content. For more information on other guidelines, see WCAG Website.

Design

1. The navigation path should be easy, consistent, and should have multiple ways to navigate. Include breadcrumbs to help users with orientation.
2. Your site should display well with different viewport sizes. For example, mobile phones and large screens.
3. Provide ample contrast between the foreground and background. A contrast ratio of 4:5:1 is recommended. Also, color should not be the only way to convey information or instructions.

(continued on page 5)

CONTENTS

Best Practices Newsletter

A publication of The Center for Information-Development Management.
710 Kipling Street, Suite 400
Denver, CO 80215
Phone: 303-232-7586
Fax: 303-232-0659
www.infomanagementcenter.com
info@infomanagementcenter.com

Publisher and CIDM Director
Dawn Stevens

Production Coordinator
Lisa Lambert

How to submit an article:
Kathy Madison
kathy.madison@comtech-serv.com

How to join the CIDM:
info@infomanagementcenter.com

Connect with us on LinkedIn

From the Director

Dawn Stevens

A Seat at the Table

Remember those big family holiday dinners when you were young? The dining room table was only for adults and a temporary solution had to be found for the children – typically a fold-out table, in another room, set with mismatched, unbreakable dishes. In my own children's case, it wasn't even a table when we were at Grandma's, but a large moving box with a table cloth draped over it. Not only were they separated from the rest of the family, they didn't even have a place to put their knees.

The only food at the children's table was that brought to you on a plate. You couldn't simply serve yourself another spoonful of mashed potatoes when your plate was empty. Instead, like the dog (who was often at least in the same room as the adults), you had to go beg for the table scraps if you wanted more.

Although the younger kids often enjoyed the freedom of the children's table, taking full advantage of the opportunity to play with their food and talk with their mouths full, the older children were typically forced into a parental role – cutting food into bite-sized pieces, mopping up spilled drinks, and policing squabbles. The older you got, the more you gazed longingly at the other room, wondering when you would finally be old enough to be given a seat at the other table, and perhaps secretly hoping that your parents would have a fallout with some of your aunts and uncles to free up space.

A seat at the adult table symbolized so many things: you could be trusted not to embarrass your parents; you held a higher status than the kids left behind; you would be privy to the secrets the adults kept from the children; and, most importantly, you were old enough to be taken seriously. Your voice would be heard. You could speak for the children, represent their interests in important conversations held at that table.

Often when you finally were promoted to the adult table, the realization kicked in that the kid's table was much more entertaining. At the adult table, you had to be on your best behavior; rules that were certainly not enforced at the kid's table were in full effect – sit up straight; elbows off the table; don't talk with your mouth full; don't wear your napkin on your head; and many, many more. Conversations at the adult table were boring, and you weren't invited to participate, even if you could have talked intelligently about the housing market or the various health issues facing your aging relatives. Your promotion to the table was likely due to space considerations or a grudging nod that it wasn't fair for you to remain with those much younger than you, rather than any kind of invitation to participate as an equal.

The parallels to this familiar holiday occurrence are clear within our professional lives. I hear constantly from technical communicators longing for a seat at the developer's table. The potential benefits are clear:

- We'd hear about product plans and changes immediately.
- We'd be able to more easily ask for information or clarification on topics we're writing.
- We'd be able to influence the direction of the product.
- We'd gain respect from the developers as equals on the team.

We believe that a seat at the developer's table will give us a forum to be heard and the opportunity to make a difference to our users' experience.

Nevertheless, we are often not invited to the table, even when we ask to be. We take offense and we feel undervalued. But re-read those "benefits" for attending—they are self-centered, focused on what we might gain from the opportunity. What's in it for the developers, and what is it going to cost them? Additional people at the table implies more time spent explaining and debating, slowing the design and decision process. It introduces more schedule conflicts to overcome (especially since very few technical writers I know have the luxury of supporting only one development team). In her Anatomy of Change Model™, Val Swisher points out that change is really about people and resistance comes largely from fear. Perhaps the lack of invitation is not so much disrespect, than the fear of change and the disruption it brings. We need to do a better job framing our request:

- ♦ With first-hand knowledge of the design decisions, we'll require less one-on-one meeting time with you to gather information.
- ♦ Review times will be shorter because our content will be more accurate from the information we glean from the table.
- ♦ Our knowledge of the users can help to prevent time-intensive rework by identifying problematic areas before time is spent in development.
- ♦ Because words are our strength, we can record and distribute all decisions and action items from the meeting saving you time.

In other words, in order to effectively advocate for a seat, we have to understand the value we bring, not the value we receive, and, of course, we need to be able to follow through on whatever promises we make to gain our seat.

If after reframing, you're still lacking the invitation, I point to Shirley Chilsom, the first African American woman in Congress, who said, "If they don't give you a seat at the table, bring a folding chair."

If it's important enough to you, compromise – find ways to get your foot in the door:

- ♦ Promise to simply be a fly on the wall until they get used to your presence.
- ♦ Ask to be present at only a designated portion of the agenda.
- ♦ Suggest that you don't need to attend every meeting, only once a month to start.
- ♦ Ask for the meetings to be recorded and follow up on what you heard via email, giving them a chance to see that your input would be useful live rather than in writing.

As you work to be invited, keep in mind that like our childhood expectations for the adult table, once at the table, our professional expectations for the developer's table are often not everything we dreamed they would be. There are rules to follow and an established pecking order. The seat does not guarantee a voice, let alone symbolize equality. However, rather than become disillusioned, perhaps we need to recalibrate our expectations and our approach. Instead of simple inclusion, are we really looking to transform the table in some way? To tear it apart and completely rebuild it into an image that we have? In that case, is it really a table we want to sit at in the first place? To continue the food analogy, is it serving something we want to eat? Perhaps we need to build our own table to which we invite the developers.

If we find we do want to sit at the table as it is currently built, we need to learn how to effectively participate within those boundaries. We have to be willing to put some brussel sprouts on our plate, even though we despise them. It is highly unlikely that everything will be sunshine and lollipops on day one. Remember, it takes time for a child to move up to the adult's table, and even more time to bring about the transition from tolerated eavesdropper to welcome participant.

"If they don't give you a seat at the table, bring a folding chair."

We must take the time to listen and observe what goes on at the table, before we speak up. We're not there to take over the dialogue, but to contribute. One of the best pieces of advice I've received is to assume that everyone at the table already knows what you know, and in some cases are more knowledgeable about the subject than you. Your job is find a way to deliver new insights and new perspectives on that existing knowledge.

If instead, we find that we want a different table, we're not without challenges. Now the onus is ours to define the objectives, the rules, the criteria for participating, and we must invite those we want to participate. Yet, instead of encountering the eager acceptance of a child who is invited to move up to the adult table, we may find reluctance to accept our invitation. It remains for us to prove, whether attendee or host, what's it in for everyone else. Ironically, we see the table as a benefit to us, but effective table management and participation requires us to focus on how others will benefit.

The bottom line is that everyone wants to be heard, to be included, to be treated as equals, whether we are children at a holiday dinner, employees in the workplace, or marginalized individuals in society. The "table" has become a symbol for that opportunity. However, we are not always good guests or good hosts. For this reason, we've chosen the table as the theme for our Best Practices conference to be held September 19-21 in Baltimore. We'll be digging into how to be good hosts for our own tables, enticing the active participation of our team at our management table and our customers at our user experience table, and how to be good guests at the developer's table and our own manager's table. I hope you can join us.

Dawn

CIDM

ConVEx ideas

JULY 25-27, 2022

information development: exploration, application, & solutions

ConVEx IDEAS offers 3 days of panel discussions on 12 top content related topics.

Choose from four 90-minute sessions/day

All sessions will be recorded and available immediately following the live event.

Find more information about **ConVEx IDEAS** and to register online, visit:

https://ideas.infomanagementcenter.com/

(continued from page 1)

4. Not everyone can use a mouse or prefers using it, so ensure your clickable buttons, text box, and other interactive elements, etc. are keyboard accessible. It is good practice to check if your interactive elements are ARIA labeled. ARIA labels are string of text that is the accessible name for a UI object.

5. One of the other elements that is frequently used on a website is a search bar. The location of the search bar should be predictable and should be keyboard accessible.

Links

1. Skip link is easy to implement and are a helpful accessibility feature. You can provide direct access to the main content through skip links. This will save a lot of time for keyboard and for screen reader users who do not have to go through multiple nested links.

2. Do not write links in capital letters. In general, capital texts are harder to read, especially for those with reading disabilities and screen readers.

3. Avoid screen tips and tool tips. They are not easily accessible by keyboard and non-mouse pointers like touch screens and eye trackers.

4. Add visual cues to inform the users to differentiate hyperlinks from text. For example, add underline or contrasting colors when links are hovered.

The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect.
— Tim Berners-Lee

Content

1. Add language tagging to indicate screen readers to switch to another language.

2. Ensure heading, tables, and other elements are properly tagged for screen readers to understand the content.

3. Do not write instructions relying solely on sensory characteristics like size, position, color, or sounds. For example, highlights in your instructions may be bold or in a color that cannot be read aloud by the screen reader.

4. If possible, avoid PDFs as it provides low user experience to users with vision impairments. If PDFs are not annotated properly with structure tags, screen readers find it difficult to interpret and can read out of order. If you still prefer a pdf, try to provide it as an alternative to webpages or other editable documents.

Visuals

1. Add alternate text to visuals. Alternate text or ALT text should be simple to understand, relates to the context, and is not heavy with keywords.

2. Add closed captions and transcripts. This not only helps a user who is visually impaired but also in conditions where you are in a noisy environment and would want to watch a video.

3. Avoid GIFs as they contain actions and visuals and are difficult to explain in ALT text. If you need GIFs, ensure they are not longer than 5 seconds and provide pause option.

4. Scalable Vector Graphics (SVG) provides many accessibility benefits to disabled users. SVG images are scalable as they can be zoomed in and resized by the reader as needed. Scaling can help users with low vision and users of some assistive technologies.

Sowmya Jayakirthi
VMware

A Staff Technical Writer by profession with an experience of 15 years in the industry. I have worked with VMware for the last eight plus years. I have experience working with mainframe technology, hardware and cloud computing services. I am keen on learning, exploring, innovating new technology to improve user experience.

Usha Rani Gunapu
VMware

I have around 13 plus years of experience in the field of Technical Communication. I am currently working as a Manager with the Information Experience team at VMware and leading a team of Technical Writers in Bangalore. My team focuses on End User Computing and Cloud Platform products. I am passionate about learning, simplification, and innovation.

Best Practices

As an organization, you can follow the listed best practices in your writing guidelines and processes to be Section 508 compliant:

- Incorporate accessibility as part of your documentation or web designing process.
- Consult with an accessibility expert before designing your websites, visuals, and documents.
- Do not rely entirely on tools for checking your website accessibility score. A combination of automatic and manual tests is recommended for effective results.
- Include resources around accessibility as part of your hiring process.
- Conduct refresher courses on accessibility for your team.
- Plan for audits periodically.

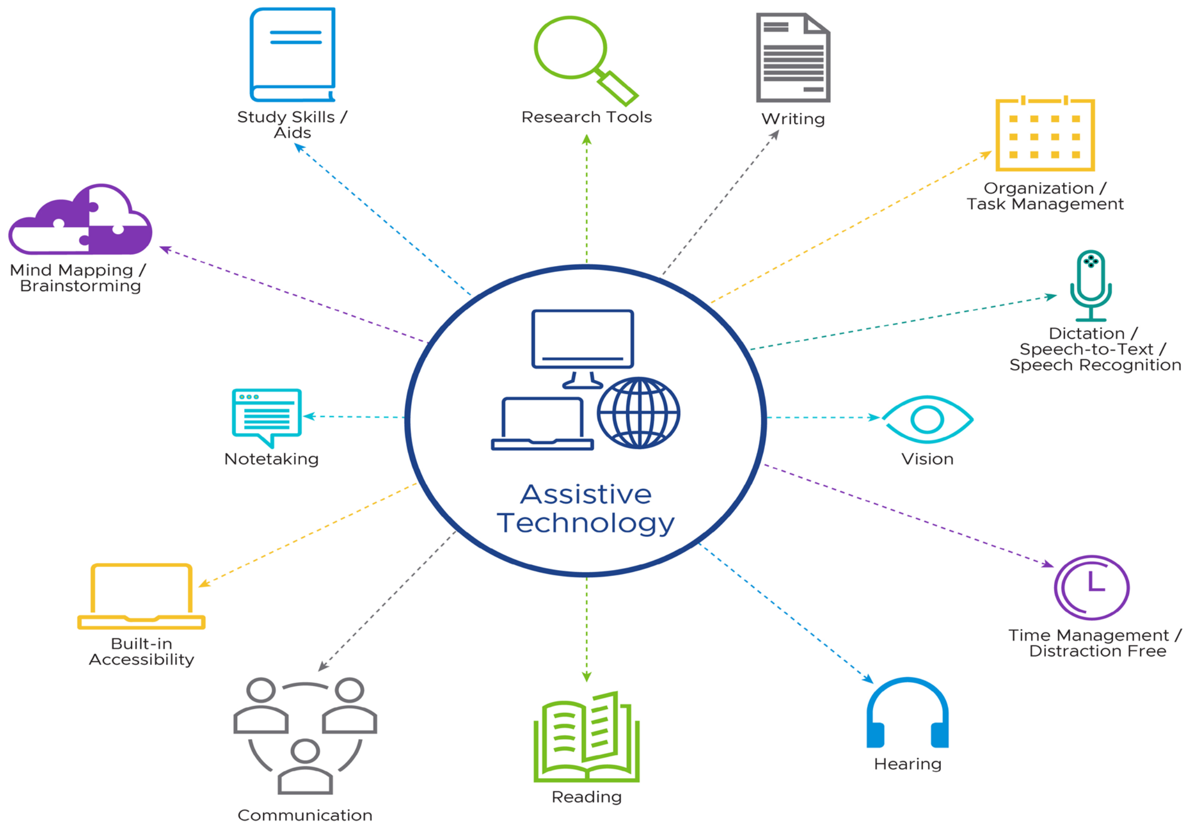
At VMware, we have been practicing the accessibility principles very closely to design our websites, documents, and graphics. We have an in-house tool that provides self-service with respect to accessibility testing.

Think Beyond

Following the best practices and guidelines from WCAG will make your websites and content accessible, however, there is no limit to how you can further enhance user experience. While we were researching how to improve user experience, we realized leveraging *assistive technology* can be one of the ways.

Assistive Technology

Assistive technology has been in the industry for many years. There are diverse types of assistive technology that can help assist the disabled community. As of today, there are many browser extensions and third-party software installers that you can integrate to your website. One of the assistive technologies that have been used extensively is the Voice Assistive technology.

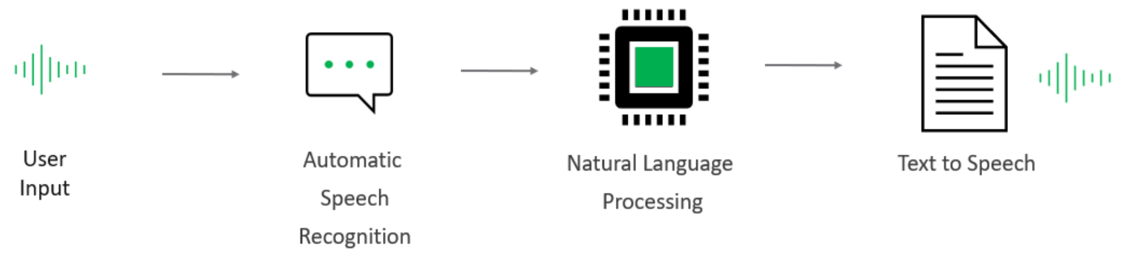

Assistive Technology
Study Skills / Aids
Research Tools
Writing
Organization / Task Management
Mind Mapping / Brainstorming
Dictation / Speech-to-Text / Speech Recognition
Notetaking
Vision
Built-in Accessibility
Time Management / Distraction Free
Communication
Reading
Hearing

During our research on Voice Assistive technology, we could not find documentation sites with voice feature integrated. We realized this is an area that is unexplored. At VMware, we have now started working on a prototype that would help us achieve this goal of integrating voice technology in the documentation websites.

Voice Assistive Technology

Voice Assistants can be used by all kinds of users and is not confined to people with disabilities. Users with crucial time crunch, users with multiple monitors (multi-tasking), and many can make use of this technology to make their lives easier.

How does Voice Assistive Technology Work?



Voice Assistive technology is a combination of Voice Recognition, Artificial Intelligence, and Natural Language Processing (NLP). The speech recognition enables the device to recognize their input and translate it from speech to text. Once the text is interpreted, the NLP identifies the intent behind what the user said, the meaning of the words, as well as the context. The machine knows what you want to search, it searches for a valid answer, and responds accordingly. So as an output, the response is converted from text to speech.

Benefits of Accessibility

- Positive impact on your business and user experience
- Caters to all kinds of users
- Reduces time
- Multitasking
- Increases website traffic
- Reduces support call
- Inclusive and relevant to all

Accessibility is soon going to be necessary for all that we design and develop. Lots of innovations and advancements are happening in the field of accessibility, especially around Voice Assistive technology. There is a need to realize and leverage the potential of the existing technologies. In future, integrating with such technologies is going to be seamless. It is time to think out-of-the-box solutions for accessibility and play a role in building an inclusive world. Let us make a difference.

DITA as code - A modern approach to the classic standard

Michal Skowron and Pawel Kowaluk, Guidewire Software

Michal Skowron
Guidewire Software

Michał Skowron has been working as a documentation specialist at big and small companies in the software development industry since the beginning of 2012. Michał is a committed proponent of automating the content delivery process and prefers smart documentation building to traditional typing. Therefore, his main focus in the recent years has been on developing and implementing tools and solutions that help organizations align their technical documentation with best practices of software development. Michał is also a board member at ITCQF and the originator and co-host of the "Tech Writer koduje" podcast.

The title of this article may raise some eyebrows among our fellow content specialists. We are aware that for some of them "DITA as code" may sound like an oxymoron or be the last thing they would think of. Nevertheless, we decided to explore this unconventional idea and we hope that you will join us on this exciting journey to discover new opportunities for the tech comm world.

Purpose of this article

This article aims to show how you can use DITA in the docs as code model. Unlike some of the more accessible but simpler markup languages like Markdown, DITA offers a structured framework for content creation. At the same time, it has enough flexibility to fit modern workflows and create a collaborative space for cross-functional teams.

We're not saying that:

- DITA is a silver bullet for your content challenges and you should **always** use it
- DITA is better than Markdown or other markup languages
- CCMSes are pure evil

Instead, we want to convince you that:

- Thinking "either DITA **or** docs as code" is not right
- Docs as code is not reserved only for light markup languages and popular static site generators
- DITA can be cool
- DITA can still hold up in the face of rapidly changing modern technologies
- DITA can be used in the docs as code model and in some cases can give you more benefits than simple markup languages

Tribute

Before we move on to the areas that we want to explore, we would like to stop for a moment and pay tribute to the pioneers of the "DITA as code" idea - "DITA For Small Teams". This project hasn't been active for a few years now but we still encourage you to learn more about it at http://www.d4st.org/

What is "docs as code"?

Docs as code isn't a specific tool or solution. It's a philosophy, approach, model of work where you use the same techniques, tools and processes for documentation as you use for code.

Typically in this model, you store sources of your documentation in a version control system, you write your content using a markup language and you automate the process of building and publishing the docs. However, there are no hard and fast rules on what tools and technologies constitute a docs-as-code setup. That's why we are bold enough to claim that DITA can work well in such a setup.

If you want to read more about docs as code, you can try these resources:

- "Docs like code" by Anne Gentle
- https://technology.blog.gov.uk/2017/08/25/why-we-use-a-docs-as-code-approach-fortechnical-documentation/
- https://www.writethedocs.org/guide/docs-as-code/
- https://www.knowledgeowl.com/home/docs-as-code

Goals of the docs as code model

The docs as code approach sounds interesting but why should you bother to use it? What benefits do you get?

Here are the main goals that docs as code tries to achieve:

- Better collaboration with developers
- Easier maintenance and faster delivery through automation
- More eagerness from developers to actively contribute to documentation - they can create content in the same context as they write code
- Higher quality of documentation, as a result of all the points above
- Cost and time savings - you use the toolset that is already available in the company therefore you don't need to buy new tools, spend time on research before buying a new solution, or worry about being on your own with tech-comm-specific tools that no dev wants to touch.

Is DITA suitable for docs as code?

We believe that all the goals outlined above can also be achieved while using DITA as the authoring standard.

These goals don't necessarily require a lightweight markup language, like Markdown. Nevertheless, docs as code and Markdown have become near synonyms. The description of docs as code on the Write The Docs page, specifically lists: "Plain Text Markup (Markdown, reStructuredText, Asciidoc)". Tom Johnson in his blog post about docs-as-code tools also states that working in plain text files is part of the docs-as-code model.

This emphasis on using plain text markup draws the line between the docs-as-code model with a light markup, and a help authoring system with a binary or proprietary format and gives us the impression that there is nothing in between.

The use of simple markup is further cemented by how a lot of dev teams add a Markdown folder to their repository and render that markdown as the documentation website. Just look at examples for Next.js and React.

Why not DITA? We think part of the problem is that **vendors "locked" DITA and DITA OT in their CCMSes**. We hear about these systems all the time in the context of DITA. We have heard it so much that we started to think that DITA == CCMS.

CCMSes are meant to be powerhouses that provide an end-to-end solution for authoring in DITA. Many companies purchased a CCMS to reduce the time required for DITA implementation. We hear stories that using DITA without a CCMS is hard or even impossible. Is it really true or is it because these CCMSes were created before certain technologies were available? These new technologies, often freely available, open new possibilities and allow for an easier adoption of any open-source technology, including DITA.

Let us restate this liberating truth: DITA is an open and free standard and nobody forces you to use it with a CCMS. The main publishing tool, DITA Open Toolkit, is a vendor-independent, open-source implementation of a static site generator for the DITA standard. It's actively developed, it follows trends, and is well documented. You can add the standard and the publishing tool to your dev workflow at no monetary or licensing cost.

As we hinted above, DITA OT is a static site generator, just like Jekyll or Hugo. Granted, it is unique because you need to know XSL to work with it, so the learning curve may be a little steeper. But it's still just an engine that transforms one format into another. And so, you are free to use DITA OT in a docs as code setup like you would Jekyll or Hugo. You can use a free source control system, a free editor, and a free CI/CD pipeline.

It's worth mentioning that DITA OT also supports Markdown as an authoring format.

Pawel Kowaluk
Guidewire Software

Pawel Kowaluk has been in technical communication since 2008. He wore the hats of technical writer, people manager, product manager, tools specialist, and consultant. He has implemented and designed content strategies for multiple organizations and ran a number of projects to implement documentation workflows and tool chains. He also bridged the gap between documentation and marketing. He is focused on analytics-based decisions in content strategies, but values people above assets. Pawel is also a board member at ITCQF and the originator and co-host of the "Tech Writer koduje" podcast.

Tools

Let's look at tools which allow you to adopt DITA in the docs as code model.

Content authoring

In a "typical" docs-as-code setup, not too much attention is devoted to this part. Since you use a simple markup language, like Markdown, you don't need a dedicated doc editor. A text editor with a plugin will do the job.

If you use DITA, it can be a little more complex. Of course, you can create DITA content in a simple text editor, but it won't get you far and your productivity will probably be lower. Authoring content in DITA requires a more robust and powerful tool. In fact, writing DITA content is more like coding, so you need something that is closer to an IDE than a text editor. A good example of an IDE which supports DITA is Oxygen XML.

However, you can also use an IDE like IntelliJ or Eclipse, and it requires very little setup. You can even use a rich code editor, like Visual Studio Code. These solutions may mean you are less productive than with a dedicated DITA editor, but they can be free, and are a lot better than a simple text editor. They also have the advantage of being well-integrated into coding environments and come equipped with tools for version control and a variety of other challenges.

Version control system (VCS)

You can use any of the VCSes available on the market. The most popular VCS among software development teams is git. You want to adopt the docs-as-code philosophy to be as close to your devs as possible, so it is likely you will use git provided by services like GitHub, Bitbucket or GitLab. Git can have a steep learning curve but it's a great tool and it's definitely worth investing your time to learn it.

Static site generator

The **bad** news is that you don't have a choice here. The only free and vendor-independent tool that enables you to transform DITA into other formats is DITA Open Toolkit. It's a golden standard and many CCMSes use this tool under the hood.

The **good** news is that it's a solid and actively-developed tool with exhaustive documentation and its maintainers try to keep pace with technology trends. For example, DITA OT offers an official Docker image and support for Markdown.

You can extend the tool by adding your own plugins. If you know XSL and Java, you can build some really powerful stuff. Just look around the official plugin registry to see what's already available. Or, if you prefer, you can use one of the existing output formats and modify it to suit your needs. One option could be to configure the DITA OT HTML5 output with CSS and JavaScript until it becomes a fully-fledged static site in its own right. Another option is to consume the HTML5 output into your existing website or web CMS and integrate seamlessly into a publication pipeline that already exists at your company.

Having just one option for selecting a generator can definitely raise some objections. If you use Markdown, you have a plethora of options. But when you decide to use restructuredText, the number of available options drops down significantly, and you have two or maybe three engines to choose from. Maybe it's some kind of mysterious law of nature - the number of available generators decreases as the complexity of the markup language increases.

Local builds

You can build DITA locally from your command line. At first, installing the DITA Open Toolkit may seem like a daunting task. In reality, it's as simple as, or even simpler than, installing a regular static site generator. For example, the installation instructions for Jekyll, one of the most popular generators, tell you that you need Ruby, RubyGems, GCC and Make. On the other hand, DITA Open Toolkit requires only Java (JRE or JDK), and maybe HTML Help Workshop if you want to generate Microsoft CHM Help. You can also install DITA OT via Homebrew or use an official Docker image.

Another option is to use transformation scenarios in your DITA IDE. For example, Oxygen XML offers this feature, and does not ask you to install anything extra.

Automatic testing

There are two major areas of testing - making sure the documentation is published successfully and making sure the content is right.

The first area is familiar to people who maintain websites. You need to make sure your content was published successfully which you can achieve by reviewing differences in snapshots between the previous version and the current version.
You also need to check whether all links work, all images display, and all accessibility and performance goals are met. Finally, you make sure that your site is discoverable by web crawlers, if that's a concern.

The second area is more familiar to technical writers. Before you publish your content, you want to make sure it meets internal standards of quality (styleguide, correct terminology, spelling and grammar). To achieve that, you can run a series of tests that will flag potential issues, or sometimes maybe even fix them.

The most popular tool to verify "XML correctness" of DITA is Schematron which can help you check how you use XML markup in your documents. You can integrate Schematron with the Oxygen XML editor, or run it at specified times, like when you push content to a git branch. Schematron is a great way to enforce your styleguide. It can check if your document follows rules like "do not create lists with one item" or "always put a path-like string of characters into a filepath tag". You can also add "quick fixes" to Schematron - small transformations that change existing content into compliant content.

To check spelling, grammar, punctuation, and other language-related issues, you can use a free program like LanguageTool, or integrate with a paid product, like Acrolinx. You could even integrate something like Grammarly with your text editor. In addition, you might want to invest in a command line tool that can measure the readability score of your documents.

It might be a little more tricky to develop something for DITA, because you would have to get rid of DITA tags, and perhaps even segment your content properly. For example, text in a `uicontrol` tag is still part of the same sentence, but an `sli` tag means a new item in a simple list. Also, some tags do not need spell checking, like `codeph` which is meant to contain code, not language.

You may have better results if you check the readability of your HTML output. The first major advantage of that is you are looking at text which is filtered by your ditaval and put together from all content references and so on. Secondly, there are a variety of tools that understand HTML, for example a tool called readability-checker on NPM.

You also want to make sure it reflects the product correctly. That last part is where DITA can truly shine because of its semantics. It gives you the power to create various tests that you wouldn't be able to use with lightweight markup languages. For example, you can test if the properties that you list in the docs actually exist in the config of your application. Or you can run the commands described in your docs and make sure they achieve the results you promise. You can see an example implementation of semantic tests here.

Content review

Since we are working with DITA as code and we are in a version control platform like Github, along with our friends, software developers, architects, and product owners, we are already part of the code review process, we just have to take advantage of it. A typical way of reviewing code is through pull requests, and we can review document sources in the same way. Github, Bitbucket, Gitlab, and other platforms come with features which allow us to comment on code, approve or reject pull requests, and prevent merging if tests are not passed.

Granted, DITA is a complex markup language which can make it harder to read than something as simple as Markdown. However, our reviewers in a software development process are people with a high level of technical sophistication, and they are usually able to read XML without any problems. DITA becomes challenging only when there are a lot of content references to parse, but this can work, as our reviewers get used to the markup and learn how DITA works.

Automatic publishing

You can use the same tools and workflows for DITA as for any other markup language. A great advantage of working in a software company is that you probably already have a CI/CD solution that you can plug your docs into. Talk to developers and devops engineers in your organization to see what options you have. Let machines do the cumbersome work of generating the output and publishing it to the server.

What about reuse?

In the docs as code philosophy, there's no place for a CCMS. However, one of the most useful aspects of a CCMS is how it helps with reuse . When you switch to a git-based solution, this feature is no longer available. So what can you do to make up for this loss?

Before we jump into devising a technical solution to this challenge, let's think about reuse itself. We often hear that it's very beneficial and gives you nothing but advantages. But the truth is that content reuse comes with challenges. You need to decide how granular your reuse should be. Is reusing topics enough? Or maybe it would be better to reuse paragraphs?

You also need to think how widespread reuse needs to be. Across one document? Across a document family? Across all documents? Each option has its pros and cons. For example, reusing content across all documents may seem like a good idea, but after some time you will realize that it's harder and harder to keep the content generic enough to fit all scenarios. Also, every change is more expensive because you need to analyze it from the perspective of every place the content is used.

Taking all these factors into account, it may be possible to address content reuse needs by simply making a smart decision when dividing content into git repos. For example, you can keep all docs belonging to the same product family in one repo. This way, you can reuse content between specific docs. In this scenario, your IDE, like Oxygen XML, will help you with renaming and moving resources without breaking stuff. Working this way is very similar to coding. You have your software project cloned locally, you use IDE to write and edit code without breaking things.

But just like with a software project, there comes a moment when you need to use an external library, that is content stored outside your project. How can you handle this requirement? The same way you would handle it in a software project - by using dependency management.

Your document is a project that requires some external resources to build properly. In case of a software project, you have libraries available in different repositories, like Artifactory or Maven, from which you download them and then use them in your code. You need to do the same for your documents. If you need to use some assets, like common topics or images, across different documents stored in separate repositories, create a place where you will publish these assets and then add them as a dependency to your document.

We use DITA OT to build the document. It's like using Gradle to build a Java project. We could write a plugin for the toolkit that downloads the assets before building the document. DITA OT offers many extension points where we can add this task.

Another option that you can consider is git submodules. You create a repository with common assets and then add this repository as a submodule to the git repository that stores source files for the document. After that, you can pull changes to the linked submodule when you pull changes for your document. This solution has one advantage over using a plugin for DITA OT - the shared resources are available at the time of editing the document.

What about link management?

Even without a CCMS, link management is an area where your "DITA IDE" shines. Similar to IDEs used for writing code, Oxygen XML offers some refactoring options that help you manage links. For example, the option for renaming a resource, like a topic, isn't limited to changing the name of the resource. It can also update all references for the resource.

An IDE can help us with managing links at editing time. On top of that, we need other mechanisms that catch invalid links at other stages of the content delivery process.

At the time of committing changes, we can use a pre-commit hook in git to run a script that automatically validates all the links in the repository and then blocks the commit if it finds any issues.

We can also create validation builds that run when you create a pull request. The policy can be configured to prevent merging changes if the validation build fails. This gives you another safety mechanism that protects you against publishing broken content. Validation builds can be part of the **Automatic testing** pipeline.

Example setup for DITA as code

Here's an example "recipe" for the toolset that you could use in the DITA as code model:

- Content authoring - Oxygen XML Author with the git plugin
- Version control system - git through Bitbucket
- Static site generator - DITA Open Toolkit with custom plugins, used in a Docker image
- CI/CD solution - TeamCity
- Testing tools - Schematron, Vale, custom validators written in Python
- Hosting solution - a Node.js server serving static HTML5 pages from an S3 bucket, Elasticsearch

We aren't in any way sponsored by or associated with the providers of these tools. We simply want to give you something practical that complements the theoretical description of the tools that we provided in the previous sections. We have experience working with these solutions, so we know that this setup works well for the DITA as code model.

DITA as code - what's the catch?

DITA as code has its advantages but that's just one side of the story. To our knowledge, so far nobody has invented a silver bullet for the content delivery process and DITA as code is no exception.

If you want devs to contribute to the documentation, they may be reluctant to use an XML standard. They are more inclined to use Markdown and you may have a hard time convincing them to change their mind. Their reluctance may in turn create a temptation to delegate all content-related tasks to tech writers; they already know the content delivery system inside-out so they are able to make changes faster and better, right?

But there's hope. At the soap! 2019 conference, Panny Luo in her talk "Content as Code: A manager's perspective", showed us how her organization managed to implement the DITA as code model and how content specialists collaborated with devs. If you have a **strong business case** for using DITA at your organization, e.g., reuse, more granular control over your content, robust semantic options - you may be able to convince all the stakeholders to play along.

A big catch of DITA as code is that you have to code a lot. Maintain your development infrastructure, localization, and publication. There are tools that can help, but you have to customize them and know how to use them.

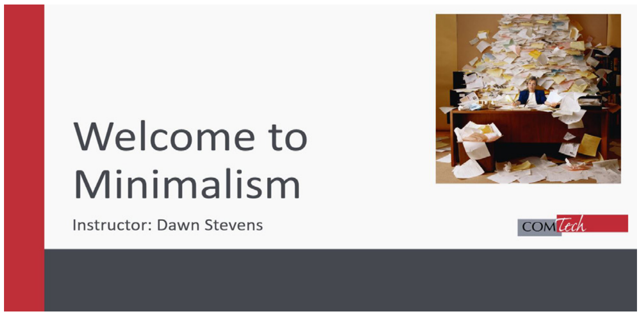
DITA as code is also harder for contributors (both writers and reviewers) who are less technical. A content management system comes with a streamlined user interface, kind of like writing Word documents. This creates a familiar experience to most people and is easy to learn for newcomers. When maintaining DITA as code, the contributors have to know the principles of VCS, understand their development pipeline, and troubleshoot daily problems with both their computers and their infrastructure.

When you write content Markdown, you can pick and choose which editor you want to use. You have plenty of options, many of them free. When you use DITA, it's quite the opposite. You only have a few options and the best ones aren't free. In a big organization this cost may not be significant but in smaller companies it can be a deal breaker.

Conclusion

Here are the key takeaways from this article:

- DITA can be used in the docs as code model - we have done it and it works.
- In some areas, DITA can offer more than lightweight markup languages, like better reuse possibilities and semantics that can be used for testing
- DITA as code is not a silver bullet and comes with challenges, like a small selection of content authoring tools and a steeper learning curve for non-technical users than Markdown
- You need to weigh all the pros and cons before deciding if the DITA as code model is the right fit for you and your team.

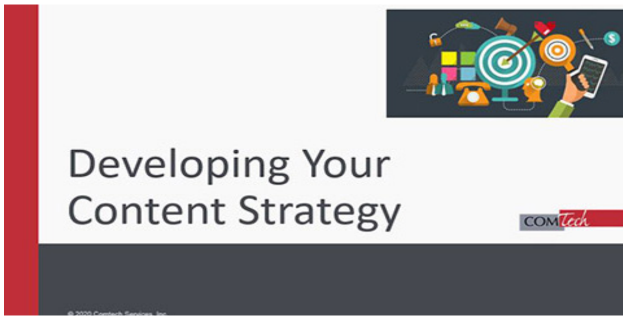

Welcome to Minimalism

Instructor: Dawn Stevens

COMTech

Two-Hour Sessions Held Weekly
Every Wednesday
September 28 — November 2, 2022

Practical application of the four principles of minimalism to select appropriate content for your users, structure it consistently, author it for easy understanding, and make it readily accessible.
https://comtech-serv.com/event/minimalism-9-22/


Developing Your Content Strategy

COMTech

Two-Hour Sessions Held Weekly
Every Thursday
September 29 — November 17, 2022

A systematic approach to defining the management, creation, production, delivery, and assessment of content, while balancing the considerations of organizational goals and capabilities with user needs and expectations.
https://comtech-serv.com/training/content-strategy/

Writing for Localization 101

Dana Aubin, Comtech Services

Dana Aubin
Comtech Services

Dana Aubin is a technical writer and content strategist based in Denver, Colorado, USA. She enjoys gluten-free baking, teaching her old dog new tricks, and karaoke spin class (virtually due to the pandemic, which is still quite fun for her, but maybe not for her spouse and dog).

Localization isn’t just about what you write. It’s also about how you write and the processes that support you. This article provides the questions that you should consider when adding translation or localization to your processes. I'll share best practices gathered from my research, conversations with industry experts, such as Dominique Trouche from WhP and Karen Ewing from Acrolinx, and organizations currently writing content for localization.

Even if you’re not localizing now, following localization best practices can improve the quality of your source content. Well-written content helps translation, but it also helps native and non-native English speakers. Additionally, being aware of the considerations and best practices for localization can help authors check their biases while writing to create more inclusive content.

Organizations consider multiple aspects when localizing their content. Depending on the maturity of the localization process, a company may do one or more of the following localization efforts:

- Language variants
- Metadata and keywords
- Taxonomy/filters
- Alt text
- URL
- Images
- Examples
- Cultural references

Translation and localization are sometimes written as T9N (translation) and L10N (localization) to shorten the terms.

Translation vs. localization

First, we need to define the terms translation vs. localization, which are often used interchangeably. Translation means changing content of one language into another language, for example, English into French. Localization, on the other hand, goes beyond translating. Localization adapts content to fit a specific market or country and adjusts it to accommodate linguistic, cultural, political, and legal differences.

If your organization is considering localization, you should start with your content strategy.

Content strategy

Content strategy is how you plan, develop, deliver, manage, and measure the types of content about your product to meet the needs of your customers throughout their journey. Your content strategy should consider localization in every part, right from the beginning.

Considerations

- What are your goals? Are you just trying to meet contractual obligations, or do you want to create content that addresses the needs of your users?
- Do you have an enterprise content strategy? If so, does it address how translated content is updated? Does it consider how you reuse content affects translation? If not, is there willingness to work together to create one?
- Do you have a corporate terminology list? A corporate taxonomy?
- What are the resources, budget, and schedule for localization? These can be the biggest factors that determine what content you can localize and to what extent you localize the content.
- What content should be localized? Into which languages and variants do you need to localize?

  * What level of localization is expected in your industry?

  * What are the legal, governmental, and regulatory requirements you must follow? Are there any political issues that you need to consider?

  * Have you conducted research on the users in your target languages? Do you have personas for your target users?
- Who creates the content—tech pubs, support, services? Does content created by all groups need the same level of translation? For some content, machine translation is acceptable, but other content needs to be translated and localized by a human.
- What tools and processes will you use to create the content and manage the translations?
- Do you need to translate any language into a pivot language (bridge language) before translating into the target language?
- Has your localization service provider (LSP) reviewed your terminology, taxonomy, information model, or style guide? If so, what kind of changes did they suggest?

Best Practices

- Have a clear idea of your goals.
- Develop an enterprise content strategy that includes localization from the beginning of each project.
- Determine what you want to achieve is feasible given your budget, resources, and schedule.
- Determine what needs to be localized; what level of localization is needed; and whether machine translation, human translation, or a combination should be used.
- Determine target languages and variants.
- Create and translate an enterprise terminology list.
- Conduct user research for your target languages.
- Choose authoring and translation tools for the enterprise if possible.
- Work with your LSP or a localization expert to ensure that your content strategy, information model, style guide, and tools are optimized for localization.

Workflows

Writing processes should include steps that improve quality because high-quality source content enables high-quality translations. Starting with good content also means less back and forth with your LSP to resolve questions and fewer changes from your in-country reviewers, which can reduce localization time.

Welcome to the DITA Publishing Workshop

Instructor: Brianna Stevens

COMtech

TWO-HOUR SESSIONS HELD WEEKLY
EVERY THURSDAY
SEPTEMBER 29 — DECEMBER 8, 2022

A hands-on walk-through of the essential DITA fundamentals, programming skills, and DITA Open Toolkit configurations required to style and publish DITA XML source.

https://comtech-serv.com/event/publishing-dita-9-2022/

Considerations

- How thorough are your review processes? Does your content go through technical review, peer editing, substantive editing, validation? Is content reviewed to ensure that content is tagged correctly?
- Do you use controlled language tools to improve quality? Writers and editors can perform quality checks before content is sent to translation using controlled language tools such as Schematron, Acrolinx, Congree, Kaleidescope, or HyperSTE.
- Do you have in-house localization experts managing the translation workflow?
- Do you work with multiple LSPs?
- Do you own your translation memory?
- When do you send content for translation? Once the English content is complete or iteratively?

Best practices

- Perform quality checks using automated tools.
- Require a thorough review process for style, tagging, technical content.
- Send for review when a subset of the final deliverable (whether topic, section, or chapter) is stable.
- In-country review by native speaker with product expertise.
- Hire linguistics professionals if you want to manage your translation memory in-house.

Structured authoring

Structured authoring is a method of writing that is organized, consistent, predictable, and reusable. Content is broken down into components, and how those components can be arranged into deliverables is defined by your information model.

Considerations

- Do you have an information model? Was localization considered when writing your information model? Are your writers trained on the information model?
- Are you using a structured authoring schema such as DITA, S1000D, or DocBook?
- Are you using a tool that supports the schema along with a component content management system (CCMS)?
- Which reuse mechanisms are you using? How granular is your reuse?

Best practices

- Use DITA as your structured authoring schema. DITA provides these additional benefits:

    * Separates authoring from formatting which enables reuse and lets authors focus on writing rather than desktop publishing. This separation also helps with languages that don't use the same formatting conventions.

    * Uses semantic tagging of elements, which can help your LSP understand the purpose of the text.

    * Provides multiple reuse mechanisms, and reuse reduces the amount of content that needs to be translated.

- Use a DITA-aware CCMS.
- Use a vendor familiar with DITA.
- Understand how your LSP localizes your content.
- Only use proper nouns as variables.
- Conditionalize sentences, paragraphs, or larger blocks of text instead of words or phrases.

Creating content

Good writing uses clear, simple, unambiguous language, and is a prerequisite to good translations.

Considerations

- Do you have a style guide?
- Are your writers using minimalism principles?
- Are you using a controlled language standard such as Simplified Technical English (STE)?

Best practices

- Develop a style guide with writing rules and a preferred word list.
- Create a terminology list with translations.
- Train writers in minimalism and localization.
- Use a controlled language standard such as STE.
- Use a controlled language tool to help enforce terminology and style guidelines
- For graphics, deliver one of the following:

    * .svg with text in the code that can be updated by your LSP

    * callouts on the image with numbers, not alphas, so that the corresponding labels can be written in the authoring tool

    * source files that can be updated by your LSP

- Include notes to your LSP if content could be ambiguous.

STYLE GUIDELINES FOR LOCALIZATION

- Write instructions as commands (imperative verb form).
- Give one command per sentence unless multiple actions occur simultaneously.
- Avoid idiomatic expressions.
- Cover one topic per paragraph.
- Use active voice vs. passive voice.
- Don't use helping verbs (for example, using "to have" to form present perfect).
- Don't use intransitive verbs as transitive verbs (for example, the window displays vs. the window is displayed).
- Use gerund form of verbs only as technical name or as a modifier in a technical name.
- Use commas after introductory phrases.
- Don't use semicolons.
- Write list items as complete sentences.
- Don't omit "that" as a conjunction between main and subordinate clauses.
- Don't omit words (contractions are ok).
- Make sure pronouns aren't ambiguous (this, that, these, those), but don't use he/she.
- Use a word for only one meaning or part of speech.
- Don't use Latin abbreviations.
- Use articles (a, an, the) and demonstrative pronouns (this, that, these, those).
- Use noun clusters of 3 words or fewer.
- Write concise and clear sentences with 20-25 words maximum.
- Write informative, not instructive notes.

SUMMARY

Best practices for translation and localization are essentially best practices for technical writing. Clear, concise writing will improve not only your translated and localized content, but all aspects of content quality while optimizing the customer experience. It will reduce the workload of reviewers and editors, decrease the turnaround time on deliverables, and reduce overall costs. These considerations and best practices for localization are the basis for creating quality content that meets the needs of your users while balancing the needs of your organization.

Editing Essentials for Writers and Editors
COMTech
© 2020 Comtech Services, Inc.

TWO-HOUR SESSIONS HELD WEEKLY
EVERY WEDNESDAY
SEPTEMBER 28 — NOVEMBER 2, 2022

The goal of technical writers and editors alike is to produce consistent, accurate, and complete information products. Reaching this standard requires a systematic approach to condensation, organization, and correction of the copy. Work through the five levels of editing and gain strategies and tips for creating cleaner content.
https://comtech-serv.com/event/editing-2/minimalism-9-22/

Please visit our web site at www.infomanagementcenter.com for more information on these and other events.

ConVEx IDEAS Conference
July 25 – 27, 2022: Online
https://ideas.infomanagementcenter.com/

GlobalLink NEXT
September 13 – 15, 2022: San Francisco, California
https://globallinknext.com/

Best Practices Conference
September 19 – 21, 2022: Baltimore, Maryland
https://bp.infomanagementcenter.com/

NORDIC TechKomm Copenhagen
September 21 – 22, 2022: Copenhagen, Denmark
https://www.nordic-techkomm.com/

Minimalism: Creating Information People Really Need
September 28 – November 2, 2022: Online Course
https://comtech-serv.com/event/minimalism-9-22/

Editing Essentials for Writers and Editors
September 28 – November 2, 2022: Online Course
https://comtech-serv.com/event/editing-2/

Developing Your Content Strategy
September 29 – November 17, 2022: Online Course
https://comtech-serv.com/event/content-strategy/

Publishing for DITA
September 29 – December 8, 2022: Online Course
https://comtech-serv.com/event/publishing-dita-9-2022/

LavaCon
October 23 – 26, 2022: New Orleans, Louisiana
https://lavacon.org

tcworld 2022
November 8 – 10, 2022: Stuttgart Germany
https://tcworldconference.tekom.de/

Best Practices 2022 is excited to be in-person again with our single-track event with a highly interactive design! The theme is **A Seat at the Table** — it's all about inclusivity, relationships, and influence!

- Giving employees a seat at your table — listening, involving, delegating
- Giving customers a seat at your table — conducting user studies, partnering with customers, and using inclusive language
- Getting a seat at your product team's table — influencing product direction, taking on new responsibilities
- Getting a seat at your management's table — being visible, getting budget

Find more information about CIDM's Best Practices Conference at
https://bp.infomanagementcenter.com/

acrolinx
Better Content. Faster.

Adobe Technical Communication

BLUESTREAM
CONTENT SOLUTIONS

componize

DCL™
Data Conversion Laboratory Inc.

Etteplan

Fluid Topics

GlobalLink™
CCMS

Heretto

IXIASOFT

KALEIDOSCOPE
Taking your content global

Quanos
Content Solutions

precision
content

RWS

stilo

[ZOOMIN]