

Creating Quality Content with Open Source Tools

By: Scott Nesbitt



When you think of Open Source software, what comes immediately to mind? Probably Linux, and maybe applications like MySQL, PHP, Apache, OpenOffice.org, and Mozilla Firefox. Applications that have entered the popular consciousness, and others which can be described as the plumbing of the Internet.

But Open Source is a whole lot more -- a wide range of utilities, applications, and tools. And a number of those tools are well-suited for creating quality content.

This presentation will introduce you to the Open Source tools and techniques that you can use to create quality content -- everything from documentation, Web content, marketing material, collateral, or whatever your deliverable is. My goal isn't to convince you to throw away the software that you're using and adopt Open Source. If you want to do that, great. I just want to open you eyes and your minds to an under appreciated and under noticed alternative to commercial software for creating

content.

A word about terminology

As you may or may not know, there is a lot of contention between the free software and Open Source software communities. They're similar, but they aren't the same. One of the main differences between the two is ideology. You can visit the Web site of the [Free Software Foundation](#) to read more [about this](#).

For the purposes of this presentation, I'm going to use the term *Open Source* to denote both free software and Open Source software. I know that's not the correct distinction to make, and that I should be using terms like FOSS (Free and Open Source Software) or FLOSS (Free/Libre and Open Source Software).

But I like to keep things simple, and for the sake of euphony and to avoid acronym overload, I'm using the term Open Source.

A little background

My adventures with Open Source documentation tools started in the early to mid 90s. In my struggling, early freelance days did a lot of work typesetting documents using TeX and LaTeX.

Around 1999, I learned basic XML and started thinking about using it to create documentation. In fact, as part of an XML course I was taking I wrote a report on how to use XML to create modular documentation. Of course, I discovered a number of Open Source tools for doing just that.

Around that time, I read [an article](#) by a Toronto-based technical writer named Michael Skeet who wrote about how he was using Open Source tools to document the Open Source software that was being developed by this then employer.

That article opened my eyes to a number of possibilities, which I'm still exploring to this day.

As a consultant, I've used various Open Source tools (some of the ones which are mentioned in this presentation) to create content for clients. I even had a hand in setting up a DocBook toolchain for one of those clients.

Quality content

The frustrated user. That's someone technical communicators don't want to see or even hear about. Quality content avoids creating frustrated users. It helps the user effectively work with a product. It informs them about it.

Quality, believe it or not, means different things to different people. One of the better definitions of quality that I've come across is from the firm that I'm doing some consulting with now: *quality is usefulness - something high quality is fit for purpose.*

As far as content goes that purpose, for me, is to inform and educate.

I wrap the concept of *quality content* in what I call the five Cs:

- Clear
- Concise
- Consistent
- Complete
- Correct

Usually, I talk about the five Cs in relation to documentation. But they apply to other content in an enterprise, too -- everything from marketing collateral to Web content to white papers. And Open Source tools are up to the job of creating that kind of content.

The five Cs have little to do with the software that you use. More on this later.

Who uses this stuff, anyway?

As you might expect, just about every Open Source project uses Open Source tools to create content.

But it's not just small projects. A number of commercial Open Source software vendors also rely on these tools to author and publish content -- not just documentation, either. Companies like:

- Red Hat
- MySQL
- Call Genie
- Amazon Web Services
- Novell (at least on the Linux side of the business)
- CodeWeavers

And a number of others. The kinds of content being produced range from user and developer guides to system administration and installation documentation and online help, to whitepapers, reports, and Web content.

Some of these firms use Open Source tools because they themselves develop Open Source software. Eating their own dog food, if you will. Others do it because they want to deal with open formats and

tools that they can fully customize. And most of those companies want to avoid vendor lock in.

That said, a number of Open Source development shops *don't* use these tools. Like who? Here are a few examples:

- SugarCRM (FrameMaker)
- Vyatta (FrameMaker)
- Knowledge Tree (Help and Manual)
- CiviCRM (Confluence)

As for why, there are a variety of reasons. You can read about some of them in [this article](#).

The tools

Which is what we're here to talk about, isn't it? There are quite a few documentation tools in the Open Source universe. Many, however, aren't flexible or robust enough for the type of work that we're doing.

The tools that I'll be talking about are editors, document formats (OK, not a tool but ...), wikis, content management systems, and tools for creating and editing graphics.

Tools that I'm not going to touch on are ones for documenting source code, for creating Web content,

and for creating knowledge base articles. While I have some familiarity with such tools, I'm not entirely comfortable talking about them.

Editors

You need to get your words down somehow. And, obviously, that's where an editor comes in. While there's no Open Source equivalent (or even something close) to a tool like FrameMaker or Blaze there are some strong and flexible editors out there.

A tool of choice among Open Source geeks, especially ones who work with XML, is the text editor. I know what you're thinking: entering XML tags manually in a text editor is painful. In the extreme. And you're right. But two editors, Emacs and [JEdit](#), have excellent support for XML. A popular extension to Emacs, called [psgml-x](#), enables you to (fairly) easily enter tags and validate your documents against a DTD.

[Eclipse](#), which can be described as an IDE for just about everything, can easily be extended to edit XML files -- including [DocBook and DITA](#). The main problem with Eclipse, though, is that it's monolithic. Eclipse can do just about everything but using it to write documentation seems to be overkill.

Which is where [Vex](#) comes in. It's basically a stripped-down version of Eclipse -- Vex has little of

the overhead of its parent. It's a lot like an XML word processor. It supports DITA and DocBook, along with XHTML, and can also use a number of Eclipse plugins.

If these kinds of editors aren't your thing, then maybe you should give OpenOffice.org Writer a look. I know what you're thinking -- not another word processor! Well, Writer is that and a lot more.

A few years back, journalist and technical writer Bruce Byfield [compared Writer and FrameMaker](#). You'd think that FrameMaker would have won hands down. Well, it wasn't that cut and dry a victory. Byfield looked at seven key areas, and FrameMaker came out ahead in only two. Writer came out on top in two, and the rest were a tie.

Writer does have some useful features for technical communicators, including conditional text and master documents that actually work. And it derives a lot of flexibility from its [extensions](#). I'll be talking more about a couple of these later.

Document formats

I think that's a more accurate description than markup languages. Many of these formats, and the software used to produce end-user content from these formats, lie in the realm of Open Source.

And there are a lot of them. Many of these languages

are what I describe as *lightweight*: they're easy to use and their conversion tools use little in the way of system resources. However, most of the lightweight formats only output HTML and aren't the best choice for documentation.

There are a handful of application languages, though, that are well suited for producing content in various formats.

I'm sure you've heard of DocBook and DITA. You might have used one or both of them, or are using them. The basic tools to transform DocBook and DITA are Open Source, and the content files can be written and edited in many applications. Some of these tools are good, but not great. More on this later.

[AurigaDoc](#) was created in true Open Source fashion: to scratch a developer's itch. This itch was to create an internal documentation system. It's an interesting format, which is a combination of XML and HTML and uses Cascading Stylesheets for formatting. The AurigaDoc engine outputs a number of formats, including HTML, four forms of online help, PDF, and Postscript. It's easy to use, and good for shorter documents.

[AsciiDoc](#) straddles the gap between lightweight formats and DocBook and DITA. It's aimed at people who write articles, reports, and short books. The source files are written using a wiki-like markup, then

converted to higher-level markup like XHTML and DocBook XML. From there, you can convert documents to PDF.

There's a [Web-based front end](#) to AsciiDoc in the works. It's supposed to let you grab individual source files, edit them, and combine those files for output. We'll have to wait and see.

Remember when I mentioned using Open Source tools to create documentation for a couple of clients? One of the formats I used was AsciiDoc. The client had a very restrictive budget, and needed an HTML-based online help system for a Web application that could be easily maintained by someone with a modicum of technical knowledge. With AsciiDoc, I created the help topics and then output them to HTML. A Perl script generated a table of contents and then everything was wrapped in a frameset. I also put together a short guide to how to update the help for the client.

Wikis

It's no secret that wikis are one of the more popular topics among technical communicators. They're easy to use, enable collaboration with other technical communicators, with developers, and with customers. They're also great for keeping track of changes to documentation and for a variety of other

tasks.

Two problems that wiki users face, though, are getting content into and out of a wiki. Sure, you can enter it directly or copy and paste content. But that's not the most efficient way of doing things. But your options are somewhat limited. There is, for example, an OpenOffice.org Writer [extension](#) that enables you to craft content in Writer and then save it to [MediaWiki](#). And, unfortunately, only MediaWiki.

Most wikis allow you to export content as HTML, XML, or even RTF. But those formats still need some tweaking to structure them. [DokuWiki](#) has an [plugin](#) that saves individual pages in OpenDocument Format. If your wiki pages have been properly marked up, the plugin will translate that into actual styles -- for example, Heading 1, Body Text, and the like.

A single tool for adding and exporting wiki content in a structured format doesn't exist. At least, not yet ...

Content management systems

Not every shop uses a CMS for their content, although many should. Open Source CMS are an interesting mix. [Alfresco](#), [KnowlegeTree](#), and [Epiware](#) aim to be the Open Source equivalents of big-name applications like Documentum. The latter two are

meant for smaller organizations, and probably couldn't handle the volume of documents that some tech pubs teams produce.

While they're actually source code control tools, Subversion and CVS are regularly used by technical publications teams for content management. There's even an OpenOffice.org Writer extension that allows you to connect to a Subversion repository and check out files. The interesting thing about this extension is that it works with Linux and Mac OS, but not with Windows.

Graphics

A picture is worth a thousand words. If nothing else, a well-designed image can help clarify a concept or an idea. There are a handful of solid Open Source applications for creating these kinds of images.

The best-known Open Source graphics tool is arguably [The GIMP](#). It's an image editor, sort of like Photoshop or Paint Shop Pro. While it's good for editing and modifying images (like screen captures), it's not so good for creating graphics like diagrams.

That's where tools like [Dia](#), [Inkscape](#), and [Xara Xtreme](#) come in. They're vector drawing programs that enable you to create all kinds of diagrams -- like flow charts, directory and network diagrams, and the

like. Each application can save files in a variety of common graphics formats, including PNG and SVG.

If a picture is worth a thousand words, then a screencast must be worth several thousand. While not quite on par with software like Captivate, you can use [CamStudio](#) and [Wink](#) to create screencasts, with and without sound.

Help authoring applications

This is one area in which Open Source falls flat. Once again, there's no Open Source equivalent to commercial tools like Flare, RoboHelp, and WebWorks Publisher. The tools that are out there are more like help authoring IDEs. You type or paste your content into an editor, and then generate your output. Hardly single sourcing.

The closest you get to single source help authoring is with DocBook and DITA. They can generate JavaHelp and Microsoft HTML Help. There's also a DocBook stylesheet for [creating browser-based help](#). You can read more about it in [an article](#) that I wrote. The problem with the latter is that you have very little control over the interface. You can't add search, additional navigation, or other controls without a lot of effort.

Why aren't there there more professional-level Open

Source help authoring tools? Partly because they aren't on the radar of many developers. And partly because coding such a tool is hard. It's really hard, and Open Source developers have other itches to scratch.

The workflow

In most ways, the workflow of an Open Source documentation project mirrors the one that you use. There are a few exceptions, though. For example, the documentation for [Xubuntu](#) (a lightweight version of Ubuntu, a flavour of Linux) and for the One Laptop Per Child project is written towards the end of the development cycle. But that doesn't mean that no documentation activities were going on before that. There is a lot of preparation and planning involved, and the writers are (of course) following the development of the products.

Many Open Source documentation projects, even for companies like the ones listed earlier, are carried out by distributed teams. Writers in different parts of the country or the world. And, some projects have the benefit of a lot of eyes on the documentation. Writers, developers, editors, QA, users. That helps catch problems.

In general, though, the Open Source documentation workflow is something like this:

- Determine the audience
- Develop a documentation plan
- Begin gathering information about the product
- Write the first draft
- Send the draft out for review
- Incorporate reviewer comments into the second draft
- Edit the second draft
- Finalize the documentation
- Have the documentation localized and translated
- Publish the documentation

Sound familiar? It should.

Adopting a hybrid solution

There's a perception that the Open Source and commercial software worlds are two very different worlds, and that never the twain shall meet. In the real world, though, you sometimes have to be pragmatic. That could mean creating a hybrid toolchain of Open Source and proprietary software.

Why? Let's take a look at a few examples. [Apache FOP](#), which is used to transform XML to PDF and similar formats, is a good tool. But it doesn't support all of the XSLT standard. When you look at a PDF produced with FOP, you'll notice widows and orphans,

problems with tables, and some spacing problems. The [DITA Open Toolkit](#) does a decent job, but I find it difficult to set up or customize. On top of that, both tools can't handle larger volumes of documents.

With DocBook (which I'm more familiar with than DITA, I have to admit), you have at least two Open Source options for better print and PDF output. One is [dblex](#), which creates PDFs by converting DocBook files to LaTeX and then typesetting them. And there's [docbook2odf](#), which is a script that converts DocBook files to OpenDocument Format (ODF) files. You can open ODF files in OpenOffice.org Writer, apply a new template, and then export it to PDF.

Still, these solutions don't come close to the quality output and the robustness of [XEP](#) or [Antenna House](#), or if FrameMaker is used as the publishing engine.

On top of that, you might want to use a more flexible and robust editor than some of the ones that were mentioned earlier. [oXygen XML Editor](#) or [XMLmind](#), for example, instead of Emacs or Vex.

A number of companies, including ones that produce and use Open Source software, use such hybrid toolchains. Novell, for example, outputs HTML versions of its Linux documentation using the Open Source [xsltproc](#) but turns to XEP to generate the PDF versions of those documents.

Going open

Now that you know a little more about Open Source tools, it's time to look at the big question: why go Open Source?

One attraction of Open Source software is [its price](#). They are free, and they can be customized. But remember that there are costs involved. More on this soon.

These tools often use simple, standard, formats. The formats are unlikely to change significantly (and won't become incompatible) in a few months or years. They're formats *everyone* in an organization can use -- no special software is needed to read or write them. There's no vendor lock in.

The software and, more importantly, the *idea* behind Open Source lend themselves to greater collaboration. This collaboration could be within your organization, or with customers. A collaborative relationship with customers is a good one to foster. Why? Customers are using what you're documenting in ways that you've probably never considered. You can improve your documentation by tapping into the experience of customers. Here's a bit of wisdom from the [Apache Lucene project](#):

I think the standard Open Source mantra is applicable here. Lucene, like most open source programs, is always willing to accept patches, especially for documentation. Furthermore, there are many excellent tutorials, PowerPoints, examples available for the basics of Lucene, many of which are available via the Lucene Wiki.

If you're a techie, or want to be one, and don't mind getting your hands dirty setting up and maintaining the toolchain then this is a good choice for you.

When *not* to go Open Source

Are there any reasons for not adopting Open Source for your documentation needs? Of course there are. Here are a few reasons for not going open.

Is your current set of tools and processes working well for you? If the answer is yes then there's no reason to switch. Too many variables may creep in -- like lost productivity to the cost of customizing XSL stylesheets -- to justify that kind of move.

You also need to consider the weaknesses of Open Source tools when you're deciding on whether or not to make the move in that direction. There's no one tool to rule them all. You'll need to roll your own toolchain from a set of disparate applications and formats. That can take a lot of time, a lot of work,

and be difficult to maintain. You just can't download a load of software off the Internet and expect them to work flawlessly out of the box (or the digital equivalent of the box). If you're not prepared to, or able, to do the maintenance and setup work then maybe Open Source software isn't for you.

If your sole motivation for switching to Open Source is to save money then I'd reconsider. A few years ago, I worked for a company whose technical publications department was, believe it or not, part of sales. We were expected to make money. Of course, the department didn't. To cut licensing and maintenance costs, they decided to roll their own solution and deliver the documentation as Eclipse Help. Management didn't think about the overhead for the writers, for testing, for development, and for the users.

Migration strategies

So, let's say you've decided to adopt Open Source software for your documentation needs. What will you need to do to migrate? Here are a few general tips.

First, consider the documentation that you currently have and how easy or difficult it will be to migrate it to an open format. It will probably involve more than just saving it in a different format and then

processing it with Open Source tools.

Next, research the tools. *In depth*. Read, download, test, and then test some more. Repeat this as often as it takes.

Consider the pros and cons of both a purely Open Source and a hybrid toolchain.

Perform the migration in small steps. Look at it as if you're inoculating yourself against snake venom. Start off slowly, with small doses of venom to build up your resistance. Take a relatively simple document and convert it. Note the problems and the pain points, and then document them. Repeat the process, using what you've learned, with the other documentation and content in your organization.

Assessing the total cost of ownership

No discussion about Open Source tools is complete without talking about total cost of ownership. Yes. TCO. Three little letters, but ones which seem to strike fear into the hearts of managers and workers everywhere. While Open Source tools are ostensibly free, there are costs involved.

What kinds of costs? The tools are free unless, of course, you decide to go with a hybrid workflow. But you may need to pay for things like training, a server that the tech pubs group uses to publish and archive

the documentation, hiring a consultant to customize your XSLT, templates, and the like.

You should also think about the time that it will take to get the tools up and running, and for the technical writers to learn how to effectively use those tools. If you're moving from proprietary formats to open ones, factor in the time that will be required to convert your existing documentation to the new format. You won't be very productive in that time.

Unfortunately, there aren't any studies or reports on the cost savings related to moving to Open Source documentation tools. At least, none that I can find. Any initial savings won't be that great, unless you're working with a huge team. You won't be hiring a new writer with those savings, at any rate.

I think that the true measure of TCO comes over the space of several years. In the short term, a tech pubs team will pick up the new tools and processes. In the longer term, your costs for licensing and product maintenance will drop. With any luck, the more technically curious on your team will be able to provide support.

The tool is not important

At least, it's not always the *most* important thing. Let's face it: the tools are a convenience for us. The

folks reading the documentation that we produce don't care if it was done in FrameMaker, with DocBook or DITA. They don't care about [single sourcing](#), topic-based authoring, or anything like that. It's not to say that those aren't important. They are, but only to us.

The people who read documentation want and need material that conforms to the five Cs that I mentioned at the beginning of this presentation.

One of the best pieces of advice on tools that I've read is this:

[T]o get the most from an XML workflow, it must be seen as much more than just a tool or a technology: there are serious organizational and cultural issues that are in many ways even more challenging than the technology itself.

That advice applies to choosing Open Source software, too.

Before you decide on your tools, you must understand your needs and the needs of your users. You just might have to rub your crystal ball a bit and try to envision how all those needs may grow. As someone said, the tools aren't the train. They're the caboose.

In the end, though, you can have Open Source software. You can have proprietary software. You can have a combination of both. The choice is yours.

Contact Us



Web site:

<http://www.dmncommunications.com>

Email:

info@dmncommunications.com

Blog:

<http://www.dmncommunications.com/weblog>

Podcast:

<http://dmn.podbean.com>