At Red Hat, we have a content services department that is about sixty people strong. Even though the department is pretty big these days, back when I started with the company, we were still trying to work out the best way to run a successful enterprise-level documentation team. What that means is that I have been involved in some of the big discussions that we have had over time about what processes we needed to get in place in order to allow us to produce the massive amounts of documentation we required as our product offerings grew. As a department, we grew very big, very fast, and our processes needed to be flexible enough to accommodate the large number of new hires we had, and still have, coming in, but robust enough to be valuable and reliable. They also need to fit in well with the engineering practices in place in the company, and the tools that our development teams use and are familiar with. Of course the other really important factor was that we had to be open. We wanted to use completely open tools to produce our docs, but we also needed to be able to work with community teams, such as the Fedora group.

Like many documentation groups, at Red Hat we use a five-phase waterfall model to produce documentation. It's based on the ever-popular JoAnn Hackos method: starting with planning, the content specification, then writing and editing, translation and production, and then a retrospective review. At Red Hat at the moment we're at a place where our development teams are increasingly using Agile-style development models to produce software, and that means the pressure has been on us to develop in a less rigid way than the old waterfall model has been allowing us to do. Also, it's no secret that the online world is changing, and people now expect to be able to interact with information at a much deeper level than ever before. They don't want to be presented with static, hard-copy books any more. They want dynamic, interactive, usable, and above all useful documentation.

In order to be able to work out what kind of model we needed to use, we needed to go back to basics. All technology is about solving problems. Back when we were sitting around in caves, we had a problem: there was all this food running around outside, but we didn't have a way to get it to stop running around, so we invented a club and solved the problem. Since then, we've used technology to solve all sorts of problems: horses were sometimes problematic to control, and they didn't go very fast, so we invented cars. The hard wheels used on early cars weren't very comfortable, and when they broke they really broke, so we invented pneumatic tyres. We also had problems being able to see in the dark so we invented electric light, being able to go to the toilet when it was raining or cold so we invented indoor plumbing, being able to send messages to people on the other side of the country so we invented the telephone, or on the other side of the world so we invented email.

Even these really technological things that we find ourselves documenting now, are all solutions to problems. One of the first things you need to be aware of when you're writing documentation is what problem your users have. If you can't describe the problem in one or two sentences, then you don't understand it well enough, and you need to keep researching. Because if you keep going, all you're going to end up with is hollow marketing spin. That's how we end up with documentation that talks about "leveraging synergies": words that sound great, but have no meaning.

So at Red Hat we came up with a fairly simple model, and that is that documentation needs to be able to be boiled down to three things:

- Describing the problem
- Solving the problem
- Giving any additional information

Anyone who has done any work with DITA would understand that what I'm really talking about here is:

- Concept
- Task
- Reference

So we've more or less said that DITA is where we need to go next. But we didn't want to completely restructure the tools we were using. We have a fairly large people investment in our tools. The main tool we use is Publican, which was developed by an engineer in our Brisbane office. It uses Docbook XML and gives us a command line interface that we can use to create new blank books, apply corporate formatting, and it integrates into our internal packaging system so we can create all these

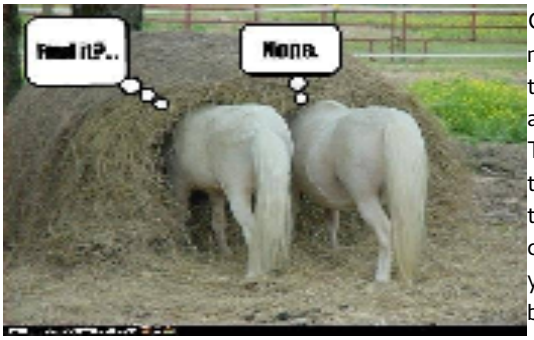
different formats for our books - HTML, PDF, and ePUB on the website, and we can also create RPM packages and man pages to package in with software. We combine Publican with SVN to give us a complete CMS, in short.

We looked at DITA and DITA-OT, the DITA Open Toolkit. We realized two things: first of all, it would take a significant amount of work for us to bring an open DITA toolchain to the level of maturity and system integration of our existing Docbook toolchain. Secondly, we wouldn't get the really significant benefits of topic-based authoring without a Component Content Management System - a CMS that has manages content at a very granular level.

Putting those two things together made it clear that if we changed to DITA all in one hit, it would take us significant time and energy just to get back to where we already were with a mature open source complete tool chain. So we decided to take an evolutionary, rather than revolutionary approach. It's a much more open source approach: to re-purpose something that you already have, add a script here, a small command-line tool there, release early, release often, and let the user community guide the development, rather than trying to design and implement some grand system in a distant (and expensive) future.

What we needed was something that worked in a similar manner to DITA, gave us content re-use and all that good stuff, but that would work with our existing Docbook XML and Publican tools. The first thing we did was to start creating topics in Docbook, using Docbook syntax, and a command line tool that we called the "Topic Tool". This was a really simple command line tool that allowed us to write XML snippets (or 'topics'), and save them in SVN. We used an extensible template model, where the topic tool retrieves a Docbook template from a central repository to match the topic type you specify. That way we can create new topic types, and even modify the Docbook syntax of existing topic types, without changing the tool on users' machines. That was an important decision, and a major part of the evolutionary "Release, Review, Refine" approach we wanted to use. Over time we did change the Docbook syntax of the basic topic types and create new topic types, validating the open source maxim "plan to throw the first one away".

The basic workflow with the Topic Tool is like this: you tell the tool which topic type you want, and it will then download the template and prefill some information for you. You can then edit the topic in a text editor, and import it back into the repository. It's then possible to view your topic from the repo directly, which means anyone can now see it and use it. We then include those snippets into any book you want using an xi:include, build the book as normal with Publican, and voila! we have a book with content reuse. So that was pretty awesome, and if you read any of our Virtualisation documentation you'll probably not know it, but that's all based on topics and maintained using the topic tool.



Of course, once we got to about 300 topics in the topic tool, we started to notice that we have another problem, we were having trouble locating topics within the repository. This made us realise that what we needed was a better way to organise it all, so we wrapped a neat interface around Topic Tool using Open Grok. OpenGrok is designed for software engineers to search source code repositories, so it worked well for what we were trying to do. This is where the open source ecosystem came into its own all over again - there are a million off-the-shelf components and projects that you can choose from to build your own system. In the end we had a web-based search tool that was pretty basic, but did the job.

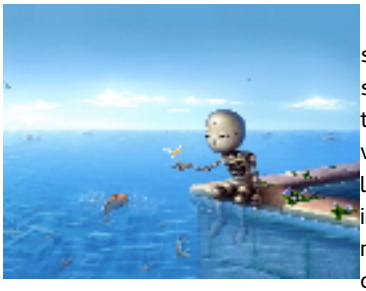
Content reuse is an obvious application of topic-based authoring, but by this stage, we'd started to realise something even more exciting. Our definition of a topic is a unit of information with a single subject - that means that it talks about one thing, and one thing only - and that has a single information role: that is, it's a concept, a task, or a reference. If we gave three topics - a concept, a task, and a reference - to a robot, along with a rule describing the "explain, answer, extra info" pattern, and some kind of graphical template, that robot can assemble those topics into meaningful and useful output for an end user. What we wanted to do was to automate this process.

When humans assemble content into a book, they are making decisions. What aspects of the information are their decisions based on, and what rules are they consciously or unconsciously using? That was what we wanted to create: A system that would allow us to store metadata about a topic, and use rules to automate assembly on a scale that we just couldn't do by hand-coding.

So we developed a system that we call Skynet, which allows us to dynamically sort and locate topics. Select the topics you want, and Skynet will download the code that presents those topics in a consumable way.

Of course, we started dreaming big after all this. We've started thinking about moving away from the documentation-as-a-book paradigm, and started considering “Documentation 2.0”. Why not include comment fields on our documentation, that will allow our reviewers - quality engineers, subject matter experts, editors, and the like to make comments directly in the book rather than creating a separate list? And why not offer that functionality to our users as well? What if we had the equivalent of a Facebook 'like' button? Users could 'like' sections that they found useful, or leave comments saying "when I tried to follow these instructions, X happened" or "this seems to be missing a step" or the like. If we break away from the book model, we start to be able to think about documentation as something that our users can interac with. We could have popular topics bubble up to the top of a list, or divide books into audiences, and present the information for each audience differently, giving them a tab to click to see the information in various ways. We could implement something similar to the Amazon “customers who bought this also bought this” and present similar topics to our readers. Using single-sourced content, and content reuse, through a system like Skynet, is going to allow us to move into these more innovative delivery methods.

The team working on the Skynet project have 110% discoverability as one of their goals, to quote the team leader: "the documentation finds you". In other words, when you're working on something, and you get stuck, the documentation is there at a click or a glance, ready for you to interact with it. Of course, I'm sure some of you are saying "Help" right now, and yes, I agree with you. That is something else we're talking about, and something that Skynet will enable us to do. Skynet pushes out XML now, and of course there's plenty we can do with that as it is, but we can also extend it to push out all manner of things, including Mallard for Gnome Help.



So let's take this conversation back to processes. All this dreaming is fantastic, but at some point we still have to actually do the hard work. Without a solid process, and a great set of standards, we're not going to be able to get there. We're doing a lot of internal testing, and we're dipping our toes in the water with the topic tool and with Skynet. So far, we've been able to slip these in to our existing standards, but that's not going to last for long. With a paradigm shift as big as this, everything is going to have to change, and that includes the way we go about producing our documentation. We need to be organised, we need to make sure what we do is repeatable, and we need to maintain our high standards of quality and accuracy in our documentation. Most of all, though, we need to maintain and even increase our focus on the customer. These changes come about not because we got bored with doing things the old way, but because we believe it's a better way to serve our audience. Never, ever forget who you're writing for, it's those poor sods out there with their problems that they're trying to solve. Our goal is to give them the tools they need to solve them.

So, to recap:

One of the main things that we have learned is that process is king. If you don't have a solid process for producing documentation, then you're going to find yourself floundering at every point along the way. You're going to end up with documentation that doesn't cover what it needs to cover, isn't accurate or well-written, and doesn't get out on time. Without a plan for how you're going to tackle the project from end to end, then you're not going to succeed. It's that simple.

The second thing is about tools. You need to decide ahead of time what tools you are going to need during the project, and make sure you have them ready and up and running before you start. It's horrible to get halfway through writing and find out that one of your writers doesn't understand how to use a semi-colon. It's even worse if you get halfway through and realise that one of your writers doesn't understand Docbook XML, or whatever authoring tool you're using.

While we're talking about tools, it's important to keep it open everywhere you can. This can seem counter-intuitive to those of you who have worked in big companies, but being open doesn't mean giving away business secrets, or exposing your competitive advantage. I think Red Hat of all companies really proves that the openness can co-exist with secure business practices.

Part of keeping it open is about keeping it real. The people behind your processes, the people doing the actual work day in and day out: they're real people. They're real people, with real lives, and real families. You need to be able to work with

people, and ensure that the loss of one person isn't going to make the whole project tumble. The other thing you need to remember is that your readers are real people as well, you need to make sure that you're giving your readers something useful, and something that they will get value out of.

And finally, I want to remind you about reviews. We all understand the importance of reviewing our writing for correctness, and reviewing our projects to make sure we can learn from our mistakes. You need to extend reviews to the documentation process itself, as well. Never be afraid to change things around. Just because it worked last time doesn't mean it's going to work next time. And just because it's worked in the past, doesn't mean it's the best way to do it in the future.