Can you measure success?

Explaining the value of documentation

Communicator

The Institute of Scientific and Technical Communicators
Summer 2012

Read the latest ISTC survey results

Practise your accessibility skills

Keep up-to-date with the latest ANSI Z535.6 standard

Using and implementing HelpServer

Global linking for flexible pattern reuse

Or how a distributed authoring team, including Owen Flatau, is using Flare and SVN to bridge the silos.

Who, why, what, and how

Within the Collaboration Infrastructure Business Unit (CIBU) of Cisco Systems, where I am one of a dozen or so technical communicators, there is a pressing need from field teams to provide solutions-based documentation.

Customers generally buy a solution that comprises a portfolio of our products, but our technical communicators are dedicated to individual product documentation.

Owing to our legacy of rapid growth and acquisition, the team's work suffers from a lack of consistency, in the usual content areas, such as style, terminology, and layout, in the processes and tools we use, and in the output types we generate.

We established a working group to standardise our working practices, around tools that we were already using, with the aim of improving consistency and thus paving the way for meeting the external need.

The working group soon realised that the central theme of our efforts would be a strategy for pattern reusability. That is, to make our authoring patterns – our processes, tools, layouts and eventually style and terminology – repeatable within the team. I use the term 'pattern reusability' to distinguish our goal from the better understood 'content reusability' because we were so diverse in the use of repositories, tools, and outputs, that we needed to rethink the surrounding content framework before we could address the content itself.

The lightweight combination of Flare and SVN was very appealing.

Essentially, we wanted to synchronise, to the best extent possible, the editing environment for everyone; we knew we would need shared product family and model names, common conditional text rules, shared templates (called 'blueprints' later on), reliable terminology, a common repository, and guidelines for keeping all of those good things in check to allow the reusable content to proliferate.

Process and environment

In retrospect, drafting this article, it may look like this was a carefully planned process; I want to dispel that nascent myth because it has actually been an organic, backburner type of project, which has rumbled on behind our daily project work, as and when we can afford the time, for over a year now. It simply is not realistic to expect a block of reserved time to be allocated to do this kind of thing properly, and none of us would have it any other way.

During the design phase we met weekly, for two to three hours at a time, to argue, catechize, digress, and assign tasks. We recorded our decisions and to-do lists in a team wiki.

We settled on MadCap Flare as the editing environment, because its global linking features are capable of delivering the goods, but we also needed a safe and accessible place to store our source. Our regional offices have separate fileserver infrastructures, so file shares were ruled out; furthermore, our sites were in the process of being brought into the Cisco production network which, at the time, was raising more questions than it answered.

We decided on using Subversion (SVN), a version control system that was already being used by technical communicators in our Oslo office. Our representative there arranged a new repository for us and mapped the process of getting accounts and installing TortoiseSVN (a client for the Subversion server).

About that time, MadCap introduced integration with SVN but our first experiences of it steered us towards the more established Tortoise option. We may go back and try again, now that the MadCap feature is more mature.

This combination, of Flare and SVN, suits our pattern reusability goal because it enables the framework and content to be edited independently, and locally, by any technical communicator in the team, yet protects the master repository for us all.

The TortoiseSVN documentation site has a great article that explains how the copy-modify-merge versioning model is different from – and perhaps more productive than – the classic lock-modify-unlock model, with which most of our team are more familiar.

So why did we eschew a more conventional approach, such as a content management system (CMS)? Simply put, we were in a state of transition, of administrative applications, processes, and tools; we all felt the inevitable drop in productivity, and we opted for a solution over which we could exercise some control. We also weren't certain about what other technical communicators in Cisco were doing, whether there were existing solutions, or whether we had time or budget to research and customise a 'proper' CMS. In this environment, the lightweight combination of Flare and SVN was very appealing.

However, none of us had done this before, not on this scale, with Flare, and we knew that there was going to be a hill to climb. I was inspired by what Thomas Bro-Rasmussen's *Single Sourcing with MadCap Flare* talk offered, and I remember distinctly how he said that although setting it

up was tricky, once done it was well worth the effort: *because it meant no-one had to remember* whether this or that version of a shared item was up-to-date or divergent.

Design considerations

With the environment in place, what things did we want to include in the shared patterns, what things needed to remain independent?

First there were the obviously common elements: page layouts, stylesheets, (Flare) templates – which is why we chose the word 'blueprints' instead – and certain boilerplate topics such as the copyright and title pages. We had few arguments and many tasks.

Then there were items that required more thought: global and local variables, condition tags, TOCs. We had moderate arguments that resulted in small, difficult, and sometimes controversial tasks.

Finally, there were aspects of our utopian vision that threatened to halt the project altogether: shared glossary, library structure, import file philosophy, checkout depth, content with multiple tags applied. Deep investigation was required and we are still arguing about some of these today.

So how would we actually do the sharing? Early in the design phase we developed a set of principles that would help ensure the veracity of common patterns without impacting the independence of our team mates who did not need to be at the shared-content table. We probably owe a lot to other Flare evangelists here, but our first principle was to collect the obviously common elements into a global 'master project'. Only one person was authorised to edit that project at first, but the number of related tasks grew and we started sharing the work.

Of course, that was only the first principle, and the implementation details would probably bore you. The more interesting principles – those about contributing, discovering, and fetching shared content – we derived from fruitful arguments and whiteboard scribbles that eventually became our 'global sharing architecture'. Figure 1 shows the centrepiece with which we beguiled our colleagues and, as we argued over it and refined it, we recorded the guiding principles that would keep it from descending into chaos, errm, democracy.

We stumbled upon a workable solution and were naturally eager to sell it to the rest of the team.

We stumbled upon a workable solution and were naturally eager to sell it to the rest of the team.

The global sharing architecture

Our global sharing architecture, shown in Figure 1, comprises three source layers and one output layer; the top two layers are the 'single source' for the working projects in the third layer. The Global assets layer is the highest

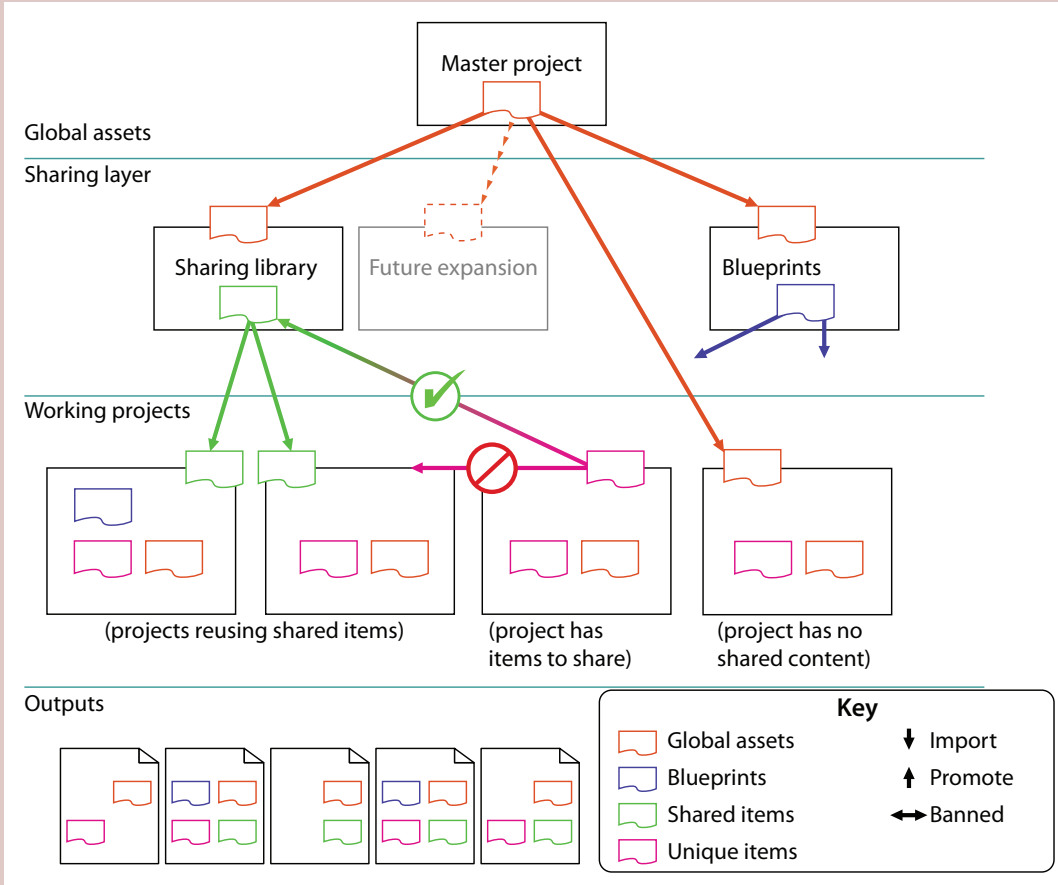

Figure 1. Global sharing architecture

The rules

1. No sideways importing: to share between working projects, promote then import.
2. Don't speculate: to be eligible for sharing, an item must be destined for at least two working projects.
3. Communicate* when promoting or editing shared content; other working projects may be affected by your changes.
   **You can use 'diff', to compare between versions but it's quicker for everyone if you communicate the essence of your changes.*
4. No spurious editing: if shared content is written in a way that seems wrong, check with the other party and editor before 'just fixing it'.
5. Avoid stagnation‡: sharing layer projects must immediately refresh global assets when the master project is refreshed.
   ‡Some stagnation in the working layer is ok: if you have imminent outputs, perhaps in review, you may not want the extra effort and risk of newly modified assets.
6. No assumptions: use conditions to keep shared content safe for everyone. Tag new content with your relevant product or product family tags so that other working projects can exclude it by default.

source; it contains only the master project which houses all the globally reusable items. Contributions to this project are well thought out and checked over with the team before being committed. It is the least volatile piece of the puzzle and all projects must import the global assets, whether directly or indirectly.

The Sharing layer contains a Blueprints project and a Sharing library project; the former for what we traditionally call templates (an empty, repeatable, working project conforming to a particular output type) and the latter for keeping reusable content. Although the layer has only two projects at present, expansion is possible if either project becomes unwieldy or if it becomes logically sensible to expand.

This layer is expected to be less volatile than the Working projects layer but will change more often than the Global assets layer.

The third layer contains the working projects from which we generate our customer-facing outputs. Golden rule: Working layer projects should not rely on each other's material. However, they must have the global assets; they may – or may not – be based on blueprints, they may promote or import shared content, or other project files, but only up to or down from the Sharing layer, not sideways, and they may or may not have unique content items.

The Outputs layer contains one or more outputs from each working project – for example in different media. Each output is unique, although they may be derived from the same content.

Of course there are setbacks, hurdles, and mudslides; it would be foolish to imagine that this project could happen without them.

Is one shared project really future-proof?

We argued long and hard about the organisation of the Sharing layer; we interviewed our colleagues about the content they expected to be able to share and discovered that we were so 'siloed' that at first we wanted geographically named folders in SVN! We had to re-examine our beliefs, think outside the whiteboard for a bit, but eventually our product families provided a sensible option. It satisfied the geography pundits, by happy accident, but also aligned closely with our branch of the metadata framework (commonly known as MDF, it is Cisco's taxonomy for product classification).

But why did we restrict it to one project? Wouldn't the content grow large, and quickly? We argued some more and decided to be more adaptable than structured: if no one could figure out what the best empty building would look like, could we arrange the foundation in such a way to make any shape of growth possible? These are the reasons that convinced us it would be possible:

- It is easy to publish an exhaustive, 'knowledge base' type of output, enabling authors to use Flare search for shared material
- There is no guessing where to point the import files
- We can cascade the global assets via the sharing layer to reduce importing complexity
- If we identify and create appropriate structure as required, the library can later be split along well-established lines (and just change or add import files in the Working projects layer)
- It is a high-impact place for our editor to focus on style, terminology, and other aspects of content consistency. Improvements here will have multiplying benefits.

What about the TortoiseSVN side of this story?

The SVN repository structure needed to change to reflect the new thinking, but we did not want to mirror the architecture. That is, we did not want all the sharing and working projects nesting in the master project folder, or *vice versa*! Our guiding light at this point was to keep the folder structure as flat as possible.

We opted to dismantle the repository silos and rebuild the root along the product family/MDF lines we had already established. The simplicity of doing this task with TortoiseSVN immediately became apparent: all we had to do was ask everyone to make sure they'd committed their own work, rearrange the folders locally, then commit the whole repository.

We put the top two layers of the global sharing architecture into one folder at the root, so that a fully recursive checkout would not be required to get the entire 'single source'. We agreed that it would be a good practice for everyone to check out at least this folder and also their own product family's root folder.

The re-organisation was over in a day. There were some tears over broken working copies, but nothing that a fresh checkout and a tissue couldn't solve.

Within each product family folder, the owning authors were given autonomy over how the organisation of their working projects was done,

provided that they honoured the no-nesting principle.

Some of our emerging best practices around the use of TortoiseSVN are:

- Update before starting work, commit before the end of the day (or more often).
- Add Flare's 'Output' and 'Analyser' folders to the ignore pattern – especially on shared projects.
- Add *meaningful* comments (even though we're forced to add comments it's too easy to be lazy).
- Branch your working project just after your outputs ship, and rename the branch with product version.
- 'Don't panic!'

Team training

Although the team may have been aware of the working group's efforts – because we had been sprinkling hints and asking for input – the emails were long and quite abstract and may have been ignored. The debate about conditional tag colours was a notable exception.

We knew we couldn't expect people to buy in to a solution with so little detail.

We chatted with colleagues in our local teams, showed them the picture and explained the principles. Perhaps surprisingly, the framework we had designed was positively received. We recorded and considered the concerns and the new ideas. No-one objected in principle to what we had planned, which was another major hurdle crossed.

We had been keeping track of our efforts, so the background work for training purposes was largely complete.

With our manager's support, we proposed a three-stage demonstration: first the TortoiseSVN basics – there was justifiably a bit of fear about this tool, being new to most; second, how to instantiate a Flare project with the master project assets, and finally, pulling TortoiseSVN and Flare together to retrieve and promote shared content from the sharing library.

We're currently about halfway through this stage, and it's going well while we continue with the remaining implementation steps.

The final piece of the training puzzle is being available to support colleagues when they encounter issues we did not anticipate (I've learned a lot about TortoiseSVN this way). We have invested heavily (in time and effort) in the success of this initiative and we want to keep our team on side as we address the remaining challenges.

Implementation progress and results

- The team has now been using TortoiseSVN, in an escalating capacity, for well over a year.
- We've been using the master project for about the same period, and have a well-established consistency in the look, the front matter, back matter, and layout of all our Flare outputs.
- The blueprints are starting to filter in, making the content and structure more predictable across some document types. This is an area we are planning to expand, as more of our legacy documentation is being migrated to Flare.
- The training phase is underway, and is appreciated by the team.
- The Sharing library and most of the thinking and implementation behind it is finished. We have yet to demonstrate it properly and start using it with purpose, but several team members are encouraging us forward impatiently: the benefits are now too tangible to ignore.

Challenges yet to be savoured

Of course there are setbacks, hurdles, and mudslides; it would be foolish to imagine that this project could happen without them.

Some of the challenges we look forward to overcoming are:

- **Different toolsets**; in some cases this is justified. As the patterns become routine though, I expect we will gain momentum. Our San Jose colleagues (within our business unit's documentation team) are going to move over to Flare, which is a good step forward.
- **Non-SVN content silos**; again, this can still be justified as we haven't yet provided the means by which to benefit from sharing content with the new framework. Geographically restricted file shares can seem a better idea than an SVN repository if there is no apparent commonality with products from other regions.
- **Terminology**; this is a bigger nut than the best practices working group can crack alone. Our team editor is hard at work developing, promoting, and integrating our shared terminology within the wider organisation. C

References

Single Sourcing with MadCap Flare www.uaconference.eu/conf2011.html#frametoflare, a talk by Thomas Bro-Rasmussen, June 2011.

TortoiseSVN documentation http://tortoisesvn.net/docs/release/TortoiseSVN_en/tsvn-basics-versioning.html, accessed 09/05/2012.

Owen Flatau MISTC is a technical communicator within the Collaboration Infrastructure Business Unit at Cisco Systems. He works closely with three colleagues formulating best practices to improve a range of product documentation and user assistance. He is a Flare fan and an avid reader of the TortoiseSVN docs.
W: www.cisco.com