Where Source Development Meets Documentation

By Elizabeth Fraley
liz.fraley@single-sourcing.com

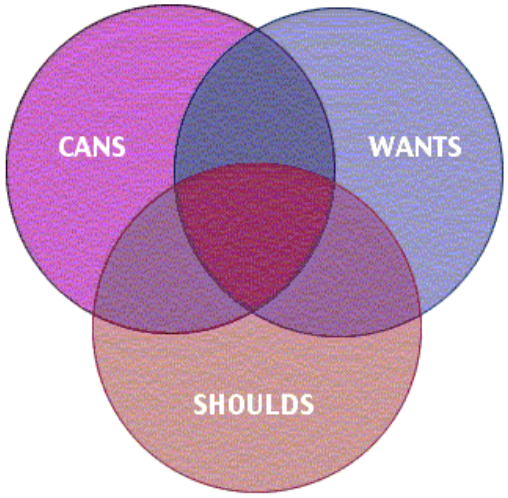
Most of the time documentation is an afterthought, if it's even a thought at all. Documentation groups are often the last ones funded, even though no product ships without it (somewhere). The truth is that more documentation shows up all the time, and not just from the company shipping the product.

User communities create tutorials and fan pages and trade information about installing, using, and extending products all the time (blogs, books, articles, wikis, and more). For a company looking to leverage all this data, single-sourcing systems are only the start. If implemented correctly, they can be extended to integrate content-generating and content-consuming organizations enterprise-wide.

I started off doing mostly XML and XSL development but have started working more in the advisory space — helping companies figure out **what** they want to do and what they **can** do. Believe it or not, I often find myself trying to explain to people why they should not pursue a project. There is this bubble mentality sometimes to go ahead and XMLize everything. This doesn't always make sense.

Consider the following Venn Diagram with three intersecting circles: CANS, WANTS, SHOULDS.

A Brief Interlude to the Real World



CAN you do it? Maybe.

Do you **WANT** to? That depends a lot on who is answering the question and why they're motivated to look at XML at all (or unmotivated, sometimes).

SHOULD you? That's the hard one. It mostly comes down to cost versus benefit.

It’s where these intersect — there in the middle — that I work, and for the remainder of this article, that’s where we are. Answering these gives you the answer to "what gets done?" and "how far do we go?" up front — and that can save people a lot of wasted effort, time, and money because it eliminates the problem of "invisible assumptions."

Efficiency is in the Tools

Originally this article was called "Efficiency is in the Tools," but that title produced an unexpected reaction. Most people thought the talk would be a product presentation or a product set recommendation (i.e., what are the "best tools" or which vendor has the "best system"). While both of these questions are important to every project, neither has a place in this article. By tool, I don’t mean product. In software development, tools are the scripts and simple programs that join larger applications together that contribute to an “efficiency of work”.

In this article, I will not endorse, promote, or recommend any particular vendor, application, or product. I have varying opinions on most of them, and my opinion changes based on the problem space being addressed. A product that is right for one customer may be totally wrong for another customer — even when both companies are producing similar products. There is no ultimate best. All you can hope for is finding the one that best fits your situation at this point in time and that provides the best foundation for the future of your project.

Situations, companies, and requirements are not static. There’s no guarantee that the requirements you have today will be the same requirements you have tomorrow, or next week, or next year. The only guarantee you really have is that all three of those things will change. No matter what system you choose — and how careful you are with your requirements when you choose it — the best you can do is to make sure that you choose a system that provides you with avenues that give you the flexibility to develop the tools that reflect your changing business needs.

This is not a new concept. Recycling content (chapters, graphics, etc.) is not new. What is new here is the common set of back-end structure in XML form and the fact that more than one set of tools — including small, mission critical custom tools — are explicitly focused on the specific needs of a given project. There is no longer any need for one tool to address all of the needs of every company that purchases a license: Single sourcing is not a tool. It is a community of tools that is easy to expand or customize without long term lock in or breaking the model.

That’s where efficiency comes from. Efficiency comes from the automation, acceleration, and merging of applications, systems, and processes that are business-specific. It lies at the edges of those same applications, systems, and processes. This is the argument that is at the core of the off-the-shelf-ware versus custom tools debate.

In a single-sourcing environment, efficiency centers around content development — source development — and its management.

What is Source Development?

Source includes any programs that a company uses in house — or integrates with. Usually we think of source as software product source code. There’s more to source code than the software product. There’s documentation in the source code. There’s feature information in the source code. All of this is source for documentation development. Where can source be found? In product source code. In the marketing materials. In the price list. It’s even in the approval chain that authorizes product release.

Sometimes source isn’t obvious. Everywhere in an organization are simple tools that you don’t even know people are using. It’s in all the little ways employees track their activities and responsibilities. It’s all the things that people do to import information, but that no one even thinks of as "source."

At one client site, a writer I knew developed a huge spreadsheet that would let him track the features that he and his team needed to document for a release. It tracked status, responsible party, due dates and engineering contacts, and other publications-specific information. He had one of these spreadsheets for every one of the five releases he was either managing or working on. In every instance, he filled out the spreadsheet by hand with tedious, strict attention to detail.

For a long time, no one knew he was doing it. Eventually, maintaining it became second nature. It provided him with an extremely accurate picture of the state of the documentation. He could respond to management's questions easily, and he was always on top of issues the minute they arose. But he was always working late, always stressed out, because managing all this information by hand was extremely time consuming, even for all the benefit the department got out of it. His experience is not unique. These kinds of tools are everywhere in your organization. And they are always manual, time-consuming systems.

Every good management structure wants to encourage employees to automate for efficiency. If you want to make people more efficient, it's extremely important to find these tools and create more efficient tools to improve employee productivity. By improving productivity and automatically collecting that information about information (metadata), you improve source generation.

Source is anything that inspects data and is dependent on it's structure. Dependencies equal time lost to invisible causes anytime something changes. Anywhere. Source is just a kind of information. It is a very specialized kind of information, with specialized uses.

Source:

- Filters and transforms information (database, statistical tools)
- Collects information (web apps, assay machines)
- Produces information (simulators, ...)

Source code is itself a product, whether internal or external. Some parts of the source code may be aimed at internal developers, end-users, or other source developers (partners). Source code can be a driver toward information exchange in others. Businesses are driven by process, but that process can change. Ideally, processes are specified. In reality, most processes are ad hoc with guidelines.

Documentation can take advantage of levels of information, the gradations of detail, in order to produce two versions of the same manual. One version of manual may include the detailed API available to engineering while another version of the same manual might leave out the huge sections of the API that are unavailable to external end users. The varying level of detail built into the source code can drive the way that the documentation is constructed.

Where Does Source Come From?

Source is created by source makers. These people create new content that stands alone. They also create the content that gets pushed back into a product. Source is created by source annotators. The product might itself have provided metadata that was the basis for the newly-created content, in either structural or template form. Annotators take chunks of generated material, shape it, and flesh it out. Source comes from every part of the organization. Anyone who is creating or annotating source can be included in efficiency initiatives.

The goal of single-sourcing is always to increase the efficiency of the entire staff as the demand for documentation increases while staffing and resources do not. The benefits for publications have been well covered in the last few years.

Single-sourcing objectives include:

- Making reuse possible
- Improving accuracy (fix once, fixed everywhere)
- Improving quality (more time per writing unit means greater quality)
- Increasing author collaboration
- Automating simultaneous delivery of source material to multiple media
- Reducing translation efforts and costs

Single-sourcing benefits include the separation of form from content. In some ways the desktop publishing revolution of the 80s could be considered a detour. In the 80s, writers for technical information ended up being given the additional task of adhering and enforcing "presentation" — something that was not their job and is not relevant to the actual task of writing. XMLized single-sourcing brings with it a return of the focus on content to writers while at the same time providing them with a much richer language for self-(content) expression. Think of "bold face" as a pidgin language way of marking something as important.

The Desktop Publishing (DTP) revolution of the 80s was a positive change. The education needed to use the applications was greatly reduced by the instantaneous visual feedback to the user and the coincident ability to experiment and explore the capabilities of the tool. It allowed the rapid production of finished documents by those who were generally unschooled in the basics of typography and layout. Writers could subsume the role of dedicated layout technicians.

This change was not free: the long term cost of the 80s-style DTP approach to document creation is that the resulting documents are by and large isolated from the larger pool of documents and the information used to create them. Even simple issues are complex: if a company logo changes, each document has to be re-worked, if they can be re-worked at all. This is often an issue when the applications evolved and lost compatibility with old document formats.

With the 80s and early 90s DTP, the productivity upside at the front end of the process is lost at the back-end as the lifetime of the content is extended and larger projects attempt to adopt the same tools. This behavior can be seen as different distributions of effort and cost: Some tools have high front-end cost but are very efficient for ongoing projects where the content will change over time and others have a very low front end cost but such a high back-end cost that the projects that use them will inevitably be restarted from scratch. It’s the difference between front/back end vs. front/back loaded.

Single-sourcing attempts to return content generation expertise to authors, return formatting expertise to format experts. It lets authors focus on correct, collaborative development. When something is added, someone knows. When something is deleted, someone knows. Tools that monitor single-sourcing systems can automatically red-flag things in process. Because individuals are freed from devoting attention to processes that tools monitor automatically, their attention is not divided or redirected away from their primary activities. As a result, documentation organizations can achieve faster development.

In some cases, tools can automatically generate content or content templates. At Juniper Networks, some volunteers from the engineering organization put together a tool that would generate the chapters for the System Log Messages book. The build process would walk through the system log messages and generate one XML chapter for every set of related log messages — straight out of the source code. Authors could drop the generated XML chapter right into the larger XML document and proceed to publishing. Authors guaranteed the readability of the log messages by editing the source code files. By working directly with the software source code, the log messages were readable by the software product users as well as the documentation readers.

How Do I Go Beyond Technical Publications?

It’s easy to get people to add something if there is no serious burden and the task is "in line" with work they’re already doing. It’s hard to get anyone developers to add anything if the work is obviously additional.

No amount of process can solve this in the real world. Water flows downhill. People do valuable "optional" work all the time. People will not work extra hard to "do it right." If you make it hard, you get less. But if you keep data local, all kinds of opportunities for integrating with other parts of your business open up to you.

How Does Enginering fit into Single-Sourcing Plans?

Auto-generate Content from Engineering Systems

A lot of content is locked up in systems that are traditionally engineering systems. What if you could auto-generate content for particular kinds of technical manuals directly from engineering systems?

Release Notes are tightly tied to bug-tracking systems. Release notes document which bugs have been fixed and which are outstanding. And for the most part, bug-tracking systems contain the basic data from which release notes are constructed. Is the bug fixed? In which release is it fixed? Is it internal-only or does it affect public-facing features? Automatically generating anything from the bug-tracking system is better than having an author manually copy-and-paste bug information on a one-by-one basis.

Request and feature tracking tools provide more information to release notes, white papers, and product specifications. A good feature-tracking tool is already tracking individual features against specific releases. In which release is this feature planned? In which release did this feature actually appear? We can automatically generate a list of features that were completed for the current release and drop the entire set of information into the larger Release Notes document.

This way, writers start with partially-complete documents not blank documents and a web-view into a database. Writers need only to polish the language, add introductory and closing text, and maybe do a little re-shuffling of items. A little bit of automation goes a long way to improving writer productivity.

Produce Documents with Varying Levels of Detail

What about documents that are 80% the same but vary in levels of detail based on the audience. Inside sales people do not need — or, in many cases, want — to know how a feature works. On the other hand, the customer service technical representative needs to know exactly how it works. Internal developers need a whole other level of detail in order to integrate the next feature.

This technique is what's called "profiling" and single-sourcing conferences are full of organizations using profiling to control translation costs. It's a short hop to consider "varying levels of detail" as just another type of translation: you're translating level of detail, rather than language, to meet various audience needs. From one source, you can create one document that is appropriate for external customers, one for internal developers, and one for field engineers.

Increase the Quality of Subject Matter Expert (SME) Reviews

Engineers struggle with change bars and large documents. More often than not, engineers get huge, isolated documents to review. They must search the entire document looking for the little bar indicators in the margin that identify documentation changes. And they must check every bar in every document in order to find the changes that apply to the work they're doing.

If you have your documents in a non-proprietary format, you can present a diff view to engineers. A diff is the output of a program that displays the differences between two text files. Normal diff output shows text that has been added, deleted, or changed. By default, lines common to both files are not shown. Lines that have moved will show up as added on their new location and as deleted on their old location. It's a compact view of difference between document versions, that gives just enough context to verify the change is correct.

This is the #1 feature engineers ask me for. Every engineer I've ever known has wanted to get diffs from technical publications rather than the usual change-bar filled document. Their time is already compressed, so the less work they have to do the better. And if they can quickly identify changes and context, they're more likely to do a review, than if they must walk an entire document manually.

Improving your reviews directly improves the quality of your documentation product for end users.

Handle API Documents More Efficiently

For software that supports API documents, there are a number of ways to completely generate simple API documents.

Sun has had JavaDoc in place for years. JavaDoc is a tool from Sun Microsystems for generating API documentation in HTML format from doc comments in source code. JavaDoc documentation is a hugely popular way of handling Java API documentation. It generates the document set at compile-time.

Software engineers write the initial documentation and authors come in to polish up the language directly inside the source code. End users are thrilled to get good JavaDoc API documents, because they're already familiar with this kind of documentation. Every JavaDoc document looks the same and divides information up the same way. Since it's part of the code, the document is nearly free to add to your product set.

Sun isn't alone with JavaDoc. Doxygen is an open source JavaDoc-like documentation system for C++, C, Java and IDL. Like JavaDoc, Doxygen generates files that can be easily compared automatically (remember about improving review activities). Both systems do one other thing: they directly tie documentation to specific API features.

Comparing two sets of generated docs not only gives you the difference for review purposes, but it will immediately identify new API features and features that have been deleted. So for documentation that is tied directly to API source code generation, you get automated realization of code differences and API changes — without having to rely on the relationship between Technical Publications and Engineering.

At the same time, because these documents are embedded in the source code, you open up another avenue for original content gathering. Domain experts can contribute directly to this model. Writers can polish and shape information that developers initially document.

Documents become combinations of XML units AND content from the software source code tree. The tight binding between source code and documentation, allows the writing staff to easily track changes in the code which are tied to changes in the documentation. It allows auto-generation of template documentation and, in some cases, complete documentation.

The Benefits Aren't Limited to Software Engineering Organizations

Only technique #4 (API documents) is really limited to software engineering organizations. These same principles have implications in other industries:

Software/Tech	Build/tools integration
Biotech	Database/assay machine integration
Pharmaceuticals	Product labeling requirements
Manufacturing	Self-service troubleshooting Integrating diagnostic information with procedural documentation
Medicine	OWL Initiatives

And what about Customer Support...

Generating Documentation from Knowledge Bases

Since we're already talking about generating content out of business systems, what about generating content out of customer support knowledge bases (KBs). Often, customer support self-service knowledge bases are populated with content that is generated by an external KB vendor. Articles with the KB are augmented by customer support representatives based on direct interaction with customers.

Obvious documentation product targets include FAQ-style documentation. But it could just as easily include common troubleshooting guide information that is based directly on KB content and the frequency of article access. Integrating with a KB system gives you the opportunity to identify which articles generate more hits — which topics do users generally need more help with — based on verifiable, measurable statistics.

If you are also tracking article effectiveness (how well does the KB article solve the user's problem), you have a pretty good basis from which to build a troubleshooting guide. Writers don't need to generate this content from scratch: they have the article to use as a tangible basis, because you generated their starting document directly from the KB in the first place.

At the same time, because you're automatically generating the initial content, you're automatically coordinating content to guarantee consistency between KB articles and documentation written from in-house publications group.

Let's not forget that you can get the links between (inside) documents and KB articles, and field alerts for free as well.

Special Benefits for Training

Training materials are perhaps the best example of repackaged data. On the one hand, training materials are perfect targets for reuse: content can come from published manuals to customer support to sales and marketing collateral. Training is the one place that you want to ensure information consistency. Customers pay for training and expect that what they learn in training will always be of value.

At the same time, the training materials themselves are a perfect example of repackaging data based on the idea of varying levels of detail. From one source, you can create scalable training materials:

- Student materials
- Instructor’s materials
- Presentation slides

In addition, with very simple tools, you to go up and down — increase or decrease — the level of detail.

For example, one of the best trainers I know teaches the skills required to do exactly this. G. Ken Holman teaches XSLT and XPath, the language that transforms XML data. He prepares one large XML document with all relevant, tagged information. From this one document, he can produce the student materials, the instructor's materials, and his presentation slides. He can switch from one to the other dynamically to address questions on the slides or in the materials, right before the students' eyes, with a simple click of the mouse.

At the same time, he can produce the individual student name plates, the registration materials, and the student contact information sheets just as quickly. He creates one large document with all required information tidbits. Because he can generate all the output files, he can process registrations and provide accurate data to the students and his class helpers right up to the last minute. One quick transformation and he can print out the latest information for distribution 5 minutes later when the class begins.

It's powerful stuff.

Training in an organization happens at a number of levels — internal, external, and management training classes. Most training classes get cancelled because not enough people sign up.

What would happen if you aggregated the schedules of all available training opportunities? You'd get a master schedule of training classes — one-stop shopping for training, if you will — that everyone could consult. That way, employees can find classes that fit into their existing schedules, reducing the amount of time that training impacts their day-to-day deliverables while at the same time reducing the overhead costs of less than full classes.

When you combine aggregation with profiling, you can generate schedules on-the-fly for different audiences simultaneously:

- This Week, This Month, Next Quarter
- Internal, External, Management
- Certification, Webinars, Tutorials, Online Training

A trainer I know does this on an inter-company basis. He gives training classes himself, and licenses his training to other people to deliver. He posts an XML version of his training schedule; his licensees post XML versions of their schedules. Each one also has a filter that pulls in the other person's information at run time.

This means that whenever someone clicks on Ken's class schedule, that person sees not only Ken's deliveries, but Laurie's too. Ken only needs to modify his own schedule: he never needs to update the master schedule. The master schedule is updated whenever a potential student clicks on Ken's schedule link: XSLT fetches Laurie's latest data, integrates it with Ken's, and displays both to the student. The customer is served: the student can always find the best class that fits his or her schedule.

It's a simple scheme that works because all business units can participate with minimal effort.

Just a note about RSS

RSS is a family of XML file formats for web syndication used by (amongst other things) news websites and weblogs. The RSS formats provide web content or summaries of web content, links to the full versions of the content, and other meta-data. In addition to facilitating syndication, RSS allows a website's frequent readers to track updates on the site using a news aggregator.

Different news and weblog sites use RSS feeds that are automatically updated whenever a new article is posted. The site determines how much information is posted to the feed — the whole article, a teaser, or just a headline. Readers can use their favorite RSS aggregator to access everything that interests them without having to visit every single site. It's a great way to keep your customers up-to-date and connected to you.

Finally: Adding Marketing, Sales, and Other Business Systems

Profiling Customer-Specific Purchasing Information

The customer price list is an obvious target for integrating with Marketing. What if you could automate creation and publication of the customer price list? A price list tends to be a complicated, dynamic document. Prices are adjusted based on the customer, the geographic location, the business segment, just to name a few. In addition, different customers may have different discount levels, based on any number of factors.

If you were to dynamically generate the price list, you could produce accurate, up-to-date focused price lists for every customer every time. Simple filters applied to the same information can produce different views. Profiled data turns a single data source into customized price lists for your customers.

In fact, taking this one step farther. Most websites require their customers log in to get pricing data. As long as you know who's looking, you can profile your way to displaying targeted pricing data: you can put their most likely purchases at the top of the list, and, depending on how good a customer they are, you can alter their discount to make purchasing more attractive.

How about the parts list? You can provide customized parts lists to your customers if you integrate their purchasing history with the parts database. Mission critical custom tools can bridge systems through the simple application of XSLT to XML data acquired through XML interfaces. You get documentation stubs for free, that an author or editor can polish and deliver directly to customers.

In addition, you can improving customer experience by integrating purchasing data and portal personalization. XML web service technologies can help coordinate information for customers. Use customer purchasing data to delivers product updates, white papers, and data sheets. The profile connects the docs, linking in field articles relevant to the products customer has ordered in the past or may be interested in today.

Separation of Form And Content

The separation of form from content is particularly important for marketing. Separating form and content makes changing look-and-feel very easy. This applies to more than just the CSS stylesheet applied to a company's web pages: this includes any PDF documentation, any text (README) type files, any presentation slides or self-service, knowledge base articles. Each one of these is content clothed in a particular look and feel that can and will change over time.

The average company changes the look and feel of it's external website every 4 or 5 years. The more tightly integrated the content is with look and feel, the harder it is to upgrade your look. At Juniper, the marketing team outsourced a new look-and-feel to a web services company. It took nearly 9 months to migrate all the pages to the new look. It took another year to convert the pages to more easily maintainable ones.

The web services company had traded maintainability for precise look-and-feel. As a result, they had little pixel gif images all over the place, embedded throughout every single page, so the page would "look perfect" every time in every browser. As it turned out, it didn't look perfect in **every browser**; it didn't look right in half of the FreeBSD-Mozilla browsers **internal** to the company. And it made updating pages, with changing content lengths extremely difficult.

On the other hand, if they'd automated page generation, updating pages would have been a breeze. Update information in a stub-page, send the stub through a generator, and boom! You've got a well-formatted page that fits the new content. What you don't have is a lot of manual effort to tweak all of the various little spacers that the web services company threw in just to get one page to look just right.

Finally, a few ways to include internal business processes

- Supports Service-Oriented Architectures
- Services to support Sarbanes Oxley activities
- Audit trails
- Branding turnover
- Acquisitions
- Connecting to operations, manufacturing, and document control systems
- HAZMAT database integration
- Universal Business Language (UBL)

Look for places that create source data that is repackaged for different audiences and different purposes. All of these places are potential targets for cost savings through XML tool development.

We're all sold on structure – that's why we're here

But we want our system to work. We want it to be able to interact in all of these ways and all the ways we haven’t yet thought of yet.

So, when you’re implementing, remember:

- Structure is added work
- If you make structure onerous or make adding it hard/expensive time-wise, people will not do it
- You need to be careful about what you choose to do or you will end up with structure that may not really be useful: <temperature>47C/104F</temperature>