Abstract—In developing new ways to publish vast amounts of information, many technical communication teams face problems that go far beyond the challenges of one book, a series of books, or even a series of CD-ROMs. Technical communicators begin to face a constellation of problems that are more like those that have plagued software development since it became a distinct profession in the 1960s. At first a project seems promising. Then, as the work begins and progresses, we become enmeshed in interlocking problems of management, purchasing, staffing, training, installation, integration, and vision. This article summarizes the lessons learned from a major effort to use the Standard Generalized Markup Language (SGML) to pull together into a single, accessible, electronic “publication” large amounts of very complicated information.

Index Terms—Department of Energy, document management, HTML, SGML.

In August 1991, the U.S. Department of Energy (DOE) announced an initiative to develop standards for the electronic exchange of information among DOE sites, and they hinted very strongly that DOE would require Standard Generalized Markup Language (SGML) as the enabling technology. DOE then established an SGML Technical Working Group to research the issues and prototype applications. Eager to comply with DOE’s intended direction and at the request of two potentially large customers, the publications department at Sandia National Laboratories invested in a full-featured, SGML-based, document management system in fall 1992. Four years later, the Environment, Safety, and Health (ES&H) Center (one of the two original customers) successfully demonstrated the prototype Computerized Document Management System (CDMS) before canceling further development and mothballing the system. For organizations contemplating their own SGML-based computerized document management system, this commentary identifies lessons learned.

Background
Sandia National Laboratories is a multi-purpose research and development facility. Because of its diverse

Commentary

—Chris Madigan,
Michael K. Silber,
and Suzanne Wilson

Manuscript received November 1996; revised January 1997.
The authors are with Sandia National Laboratories, Albuquerque, NM 87185-1041 USA.
IEEE: S 0361-1434(97)04359-2.
operations, Sandia follows many environment, safety, and health laws, regulations, and directives from federal, state, and local government agencies. Almost all of those agencies audit Sandia’s ES&H programs frequently, and the documents describing those programs number in the thousands. CDMS intended to provide to the most rigorously audited ES&H programs the best tools for documenting compliance with those laws, regulations, directives, and industry standards.

To do that, CDMS associated within a database the individual passages from requirements documents (like federal laws) with implementing passages from Sandia documents (like standard operating procedures). CDMS distributed both the implementing documents and the database reports in hardcopy and electronic copy via HTML (hypertext make-up language) on an internal Web and intended to control changes to implementing documents as the requirements changed.

In the beginning, the system scope included more than ES&H documents, but development setbacks led the other major customer and the publications group to transfer all system assets to ES&H.

Challenges tended to fall into four categories: technology challenges, vendor challenges, corporate challenges, and team challenges.

**Technology Challenges**

First, learning Standard Generalized Markup Language is difficult. While its benefits are easily grasped—enter data once and use it often, in a variety of applications and output media, over the lifecycle of the document or of the product or process that the document supports—the means for achieving those results are anything but easy.

Rapid obsolescence and leapfrogging technology do not help. Within our project’s life, only one of the five major applications survived. The majority were replaced not just by a later version but by another product. The most volatile example was the electronic browsing technology. One product’s developer declared bankruptcy, and the replacement product was eclipsed by HTML browsers. And, at the time, Sandia had no successful installations on which to model its efforts, so we were stumbling to find our way.

**Vendor Challenges**

In a volatile market, vendors go out of business or redirect their efforts. Immediately after acceptance tests, the integrating vendor abandoned the market, and a second vendor did not secure development rights for another eight months. Newly developed software did not arrive on time and often lacked expected features. Beta release upgrades did not accept data the previous beta version manipulated with ease, so test data had to be reimported, thus requiring more time.

The vendor changed platforms to anticipate the market. The database interface ordered for Unix arrived, instead, for NT, with the Unix version expected a year later. Rather than wait, Sandia switched.

Upgrades providing needed features coincidentally required more powerful hardware, thus adding costs.

**Corporate Challenges**

Our organizations proved as volatile as the markets we inhabit. Customers changed. One of the two major internal customers, understandably impatient with the delay, decided to make rather than buy its publications support.

Management changed. The vendor restructured its lines of business. The company holding Sandia’s management contract changed. Sandia managers who had supported the system left for assignments in other areas. Delays occurred as departments waited to learn their new bosses’ priorities and workers were reassigned.

Priorities changed with management. New Sandia managers preferred networked Macintoshes to a high-end system, so the publications department turned all system assets over to ES&H, which meant the project team changed. One survivor compared the effort to “changing a tire on a moving car.”

While markets, managers, and team members change quickly, corporate culture changes sloooowly. As budgets tightened, customers and managers were equally hesitant to embrace a system that seemed more centralized than they were used to, requiring more interdependence than they were used to.

**Team Challenges**

As priorities changed and budgets shrank, so did the team. By the time of the prototype demonstration, seven team members had rotated in and out of the development team, including the original MIS-trained project leader. Team members who could not cover multiple specialties gave way to members who could. The final development team consisted of three people: a team leader/marketer/budget analyst, a database manager/systems administrator/programmer, and an SGML/HTML/publishing specialist.

**Lessons Learned**

Our prototype successfully displayed about 80% of the initial requirements, but that was not enough to secure funds for deployment and steady-state operation. Now enjoying 20/20 hindsight, we can say what we would never do again and what we hope we would do better.

**What We Hope We Would Never Do Again**

We would never again buy vaporware, would not even talk to vendors about products “in development.” Next time, we would want to see the software at work, in installations like ours, on hardware and
software platforms like ours. We would avoid betas and release 1.0 and resist the very strong temptation to “shape the product” or “get in on the ground floor,” regardless of the depth of the financial discount.

We would not write a purchase contract without penalty clauses for nonperformance.

We would not accept assurances that vendor personnel are experienced project managers. Instead, we would want documented project management credentials and would probably ask to see project plans for successful installations. We would not accept, “You don’t need to know about that. We’ll take care of it.” We would not believe the vendor’s schedule, nor our own. Delays happen.

**What We Hope We Would Do Better** We did much of the following but should have done it more rigorously. Hindsight tells us how we would manage the next project better, buy smarter, staff and train smarter, install and integrate smarter, and promote the project smarter.

**Manage better:** Read your company culture carefully before you consider investing. There is no doubt that SGML is technologically sexy and offers considerable potential benefits. But not every organizational culture can nurture SGML. A full-featured SGML system could flourish in a company that

- processes a large number (thousands) of documents
- that can be categorized into a handful of well-defined document types
- by a stable (not matrixed), centralized group of innovative editors, programmers, and other specialists
- according to standard operating procedures for information management and document control
- and whose funding cycle is longer than one year.

If that is not your company, consider whether your upper management could effect such a transformation. If not, think twice before you propose an SGML solution.

Hire an experienced MIS-trained project manager, and tie his or her career to successful development of the system.

Write down your requirements so they do not change every time you hear a different person describe them. And get a manager to approve them.

Scope the job small. Aim for a Volkswagen, not a Cadillac system. Satisfy real needs that you have data (not anecdote) to support. Plan for small successes (inchstones) as well as big successes (milestones).

If you choose the cutting edge, expect to bleed.

Plan the work, yet be sensitive to team members who consider planning just so much useless overhead. A strategic plan should remind you who you are. A project plan at a fairly abstract level should remind you what you are doing. Weekly goals should identify what you intend to accomplish in the short term. If everyone does not have a single script to sing from, the project drifts.

Work the plan. That means 1) follow the plan and 2) revise it as priorities change. They will.

Scale back if necessary. Most system development projects come in late, over budget, and meeting fewer requirements than intended [1], [2]. Rather than fit that description, rescope the project.

Plan to marry your vendor or at least cohabit. Do not consider the relationship a short-term affair. And do not assume you can change them after the marriage.

Love the process because products will be slow in coming.

Celebrate frequently. If you cannot find a reason to celebrate, make one up.

Be prepared to change management styles as the project evolves. Consensus during development may evolve into more directed, constrained operations as you move into production.

Have faith in your team and prepare to forgive each other frequently.

**Buy smarter:** Develop an excellent relationship with your buyer, and listen to that person. Ask the buyer’s help in specifying nonperformance penalties. If the vendor does not perform, the buyer may be the only person who can legally apply pressure.

Shop around. Do not be afraid not to buy, even if you are spending year-end money. Better give the money back than bring in a flop.

Stick to proven technologies if budget and schedule are truly important. If you can, avoid proprietary software that can only be supported by one vendor and not by third parties.

Prefer software that has been developed for SGML rather than adapted to SGML.

If possible, talk to sites using the software you are considering. If possible, visit those sites without a vendor’s sales representative. Do not depend only on vendor demonstrations.

When feasible, ask to see your data as output from a vendor’s system as part of their proposal.

Build acceptance tests into the contract. When you test, use your data, not a vendor’s data.

Ask the vendor to identify the number of full time staff who support their product. Do not buy a product that is undersupported.
If the integrating vendor leaves the market, ask for all your money back, even if your managers would reallocate it to projects other than yours.

Staff and train smarter: Train one or two key people in the technologies so they can write the statement of work for the buyer and evaluate proposals and know when they are seeing smoke and mirrors.

Let these people research the jobs that must be done for implementation. Then identify the skill sets, and hire for that skill set. Do not tolerate and train nonprofessionals just because they are available. That leads to lose/lose situations and bad feelings. You cannot force-fit people to as complex a job as an SGML system development. You would not take a totalled car for repair to a high school auto mechanics class and wait for them to “ramp up.” Do not do it on the job.

Until the SGML market matures, plan to spend $8000–10,000 per year per person to train your key people.

Install and integrate smarter: Because the system is yours and not the vendor’s, the overall project planning and control responsibility should be yours and not the vendor’s.

Phase the development in ways that make sense to your needs and in ways that display some early successes. If you work all features simultaneously, you risk being unable to isolate faulty software and interfaces, and you delay successes. Prefer frequent, small successes to a long-coming blockbuster.

Balance change control with flexibility. On the one hand, you need to control changes rigorously to requirements, software, prototype data files, and especially the document type definition, the “DTD.” If you do not control requirements, you will constantly reach for a carrot just beyond your reach, a feature that some-one you’d like to impress wants this week. (Next week, the feature changes.) New software versions require reworked interfaces and renewed testing. New sets of data may require DTD revision, and in SGML systems using many software applications, almost everything depends on a stable DTD.

On the other hand, rigorous change control is like cotton in your ears. While it enables you to ignore the siren’s song of upgrades and improved functionality for its own sake, it may also deafen you to the roar of a waterfall you may be approaching. If development takes considerable time, periodically assess whether your customers still need what you are selling and whether the technology has evolved in ways you had not anticipated. Regardless of your investment of funds and effort, some changes will be necessary (so reserve some contingency funds).

Promote smarter: Conceptually, conceive and promote a vision worth dying for. Operationally, start small, fill an immediate and documented customer need, and grow the system in stages as long as funding lasts.

If you must plan a multiyear project, enlist strong support from management in the form of a multiyear budget commitment in writing.

Know your limits and do not be easily persuaded to exceed them, especially with budget. Do not promise to complete the job for $200K if you need $300K. Do not promise features to powerful stakeholders that you know are not among your requirements if you have not already figured out how you would implement their nice-to-haves.

A risky strategy but possibly worth pursuing: Given the popularity of HTML, consider selling HTML first and SGML later if you can generate enough enthusiasm for tagged information to allow you the time and money to make the leap. But consider that HTML is to SGML like an abridged dictionary is to a language grammar. HTML is a very limited vocabulary for displaying data on-line. SGML is a set of rules for creating a vocabulary and using it to define data content and data structure and display it on-line or in other media.

Because the tagged files may look similar or emerge from the same editing software or be browsed on the same Web does not mean the enabling technology is similarly easy to develop. An infant can use some words to get its limited needs met, but it takes time to mature a speaker who knows a language well enough to meet varied needs. If you cannot secure that investment of time, money, and effort, do not aim for SGML.

Conclusion
They say an education is what you get when you don’t get what you want. Because HTML eclipsed SGML, the Department of Energy decided not to require SGML for the electronic exchange of information among DOE sites. And because management priorities changed, our project halted after we demonstrated the prototype. But we feel educated in SGML. We have enjoyed developing an enormously complex system, improved our skills in C++, PERL, and object-oriented database systems, and collaborated with counterparts in vendor organizations and other facilities to solve interesting problems. We did not get what we wanted, but we hope these lessons learned will enable someone else to get what they want.

Acknowledgment
Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.
References


Chris Madigan is an internal communications consultant for Sandia National Laboratories in Albuquerque, NM. Dr. Madigan has taught writing with computers at universities and national writing project sites, managed document production teams, and now coordinates communications projects in environment, safety, and health. He is a certified Project Management Professional.

Michael K. Silber is a senior analyst for RE/SPEC, Inc. in Albuquerque, NM, specializing in database management on Unix and PC platforms. An engineer with a Master’s degree in International Management, Mr. Silber has worked for McDonnell Douglas and U.S. Steel.

Suzanne Wilson is an SGML specialist for Tech Reps, Inc. in Albuquerque, NM. Ms. Wilson has developed training materials and document standards, evaluated and documented hardware and software, translated scientific articles, and managed data conversions in the United States and Europe.