chapter 2

What Kind of Thing Am I Creating?

Goodbye Documents, Hello Objects **38**

Building Informative Objects **51**

Goodbye Documents, Hello Objects

Every time we don't post product information on our web site, our customers give us grief. They want to know everything we know about our products.

—Phil Gibson, National Semiconductor, quoted in Seybold and Marshak, *Customer.com*

All the information about your company, your products, and your services will need to be XML tagged and encoded with searchable attributes so that people and software programs can selectively retrieve only the most relevant and useful information and products for their purpose.

—Patricia Seybold, Ronni Marshak, and Jeffrey Lewis, *The Customer Revolution*

Shifting from paper to the Web makes you think more seriously about the structure of what you write. You ask yourself more consciously, "What is the form I am going to fill?"

Traditionally, we wrote whole documents following loosely defined but fairly conventional structures. Different sections may have addressed different groups and served different purposes, but all those sections were bound together to form a single unit—a book, a manual, a data sheet.

But when we post hundreds or thousands of these documents on a Web site, our software often has trouble locating the particular passage a person wants to read. To help software locate specific chunks within these documents, so people do not have to read through a lot of irrelevant pages, we now identify each part of the document as a distinct object, tagging each element we write, labeling this one "a procedure," and that one "a product description."

But in a giant pile of information, tags are not enough. We must also organize each document, and each component object, according to a clearly defined structural model, so that the software can anticipate exactly where each element will appear and skip all the intervening material.

Gradually, our idea of the document is being decomposed into its constituent objects, and the document itself comes to be regarded as just another object, containing other objects nested within, like painted wooden dolls within dolls, endlessly.

As we write in this environment, our method of discovering and arranging material becomes more routinized, more conventional, and less improvisational. The group we work with sets up a model structure for each type of informative object, defining which pieces are required or optional, in what order, and we have to write the objects that will fit, like blocks, into that architecture.

But each class of object has a purpose—to answer a particular

type of question, deal with a common request, or satisfy a fairly widespread need, wish, or whim. The abstract structure reflects the team's best idea of a satisfactory response. And each new object we create following this model is designed to serve as a virtual reply to our visitors in this odd conversation that takes place over the Net and through the screen.

Arising, then, as a way of coping with the huge amount of information accumulating on the Web, this new way to come up with ideas, explore topics, and organize our writing necessarily focuses our attention on the individual building blocks that software assembles into Web pages, customizing them for niche audiences, or personalizing them for individual visitors. Subtly nudged by XML and an object orientation, by customization and personalization, and by the sheer volume of content, we are being led to worship at the altar of structure.

The problems of content

As Web sites expand in size and more people in every organization pour content onto the sites, new problems emerge:

- **Inconsistent structure and format.** A series of documents, posted as individual pages, turn out to be organized differently (because they were written by different people at different times for different purposes, but now appear together on the site). Visitors who get used to product descriptions that start with a *challenge*, and go to a *solution*, followed by a section of *features* and *benefits* may be a bit puzzled to find another product description that only offers a list of *features*. From department to department, the structure of similar pages, the layout, and the interface all subtly change, challenging and sometimes completely frustrating visitors.
- **Manual processes**. The old system of hand coding HTML tags—dropping in the standard elements of an interface and publishing one page at a time—just won't work any more when a company publishes thousands or hundreds of thousands of pages a month. The process has to be automated.

The World Wide Web is a medium that gained acceptance where earlier attempts had failed by providing the right combination of simplicity and fault tolerance. Now it faces the job of reinventing itself as a scalable, industrial-strength infrastructure strong enough to carry both human communication and electronic commerce into the new century. The story of XML and its companion standards is the story of that reinvention.

—Jon Bosak, XML Architect, quoted by Charles Goldfarb and Paul Prescod, *The XML Handbook*

- **Gigantic chunks of information, instead of pinpointed answers.** When visitors search for a specific fact, they often receive a 20-page report or a 300-page manual, and the site says, in effect, "It's in here somewhere. Good luck." That approach was bad enough when users got actual books, but on the Web, it's crazy. People need fast access to a particular fact, and just that fact, even if it appears in a tiny paragraph, sentence, or phrase. You must be able to serve up the smallest chunks of information someone might want to look at.
- **Lack of customization**. When everything appears inside large documents, it's difficult to create different content for different visitors. You need to offer parts of these documents in a new order, adding a few elements tailored just for that niche audience. But taking apart an existing whole and rebuilding it can be tedious if you work in word-processing software. On the other hand, if you can treat the pieces as objects in a database, then you can issue a wide variety of reports by picking and choosing different components.
- **The audience of "IT."** Humans are reading your stuff, but software must read it, too, manipulating the content, transforming the structure, and adjusting the format to deliver a customized version to a particular user. You may want to be able to send down a Java applet, for instance, that offers to re-order the list of book titles by date, by author, by price, or whatever—on the client's machine, without having to go back to the server and ask the original database to perform this chore. To aid the software, your text must contain tags indicating what each component is, preferably in eXtensible Markup Language (XML) with a Document Type Definition or schema to tell the software how each element fits into the overall structure. That way, the software can quickly tell which paragraph contains a date, which contains an author's name, and then—if the user asks for a list sorted chronologically, or by author—reorder the paragraphs. You are now writing for two audiences, one made of code, and the other of flesh—software and wetware.

HTML—the HyperText Markup Language—made the Web the world's library. Now its sibling, XML—the extensible Markup Language—is making the Web the world's commercial and financial hub.

—Charles Goldfarb and Paul Prescod, *The XML Handbook*

These large problems are forcing content creators to take apart entire documents, breaking them down into their components, and building new assemblages out of those chunks on the fly, to offer personalized, updated, consistently organized content through software rather than human control. To identify the chunks, we are being forced to expand the tags with which we mark up our text.

Mark up that text!

Markup is not scary. Markup is at least two millennia old. Text itself started out as a set of marks on a surface, so as the surfaces and tools changed, people naturally added extra marks that told the reader something useful about the text. This meta-information, at first, was fairly crude. For instance, in Roman days, most scrolling text was an unending stream of characters, like this:

> armavirumquecanotroiaequiprimusaboris
> Isingofarmsandthemanwhofirstfromtheshoresoftroy

Then someone invented white space, marking the division between words with an empty slot.

> I sing of arms and the man who first from the shores
> of Troy

Someone else thought of marking the beginning and end of sentences. Soon capital letters and periods set off sentences.

> Easy is the path leading down to hell, but long and
> hard the climb out.

Then came a triumph of markup—paragraphing.

> Easy is the path leading down to hell, but long and
> hard the climb out.

> The ghost led me down past the caves and the smoke,
> to the river Styx. Charon, the boatman of the dead,
> poled out of the mist.

Headings, too, came along as another way of adding information about a chunk of text, saying, "Hey, this is what this section is about."

The Song of Solomon

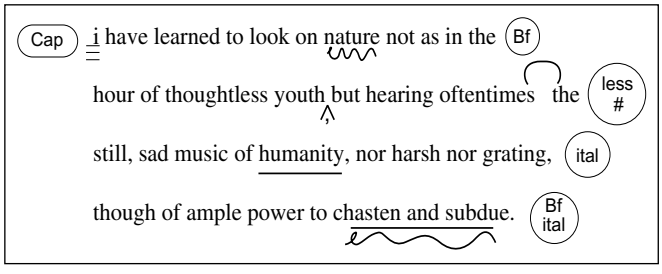
Book One

In fact, many of the formats we take for granted were invented to give information about the passage. For instance, bulleted lists could be considered a kind of markup, indicating that the items all belong together.

The commons belong to the people.

- They may run their cattle on the green.
- They may use the green for their dances.
- No man may fence in the commons.
- All souls tend the green equally.

When printing became established, we could plan changes in emphasis, weight, width, size, slant. To signal these to the printer, someone had to mark up the text, indicating what formats to use where, using editorial markup symbols and abbreviations.



> I have learned to look on **nature** not as in the hour of thoughtless youth, but hearing oftentimes the still, sad music of *humanity*, nor harsh nor grating, though of ample power to ***chasten and subdue***.

When 19th century printers came up with typesetting machines that poured lead into molds, generating hot chunks of metal that could be dropped into a tray to print with, the operators used levers to shift from boldface to plain text and back, following the editor's markup. In the late 1960s and early 1970s, when programmers were inventing word processing, they looked at those levers and decided to insert corresponding tags into the stream of characters. Each tag corresponded to a lever that the operators had pulled, when they were generating hot lead. Of course, the result was now cold type.

> I have learned to look on <bf>nature</bf> not as in the hour of thoughtless youth, but hearing oftentimes the still, sad music of humanity, nor harsh nor grating, though of ample power to <bf>chasten and subdue </bf>.

Each word processing machine and each program had its own proprietary set of tags with different tags for each element of markup. Boldface might be indicated by [b] on a document coming from one vendor, {bold} in a document from another vendor, and #—emph—# in a third. But organizations such as airplane and tank manufacturers that had to bring together hundreds or thousands of documents from different word processors ran into a problem. They had to pay to have some programmer convert all the different tags into a single tag system, or vocabulary, so that a single printer could run the job. Eventually, the Pentagon and its suppliers asked computer vendors, publishers, and librarians to come up with a solution.

Working at IBM, a team came up with a new approach. They saw that book designers typically analyzed a manuscript looking for the major components and diagnosing the structural relationships between them (this is a major heading; that is a minor heading, and

If a tag identifies data, that data becomes available for other tasks. A software program can be designed to extract just the information that it needs, perhaps join it with data from another source, and finally output the resulting combination in another form for another purpose.

—Elizabeth Castro,
XML for the World Wide Web

this is just running text). Assuming that most documents had some kind of hierarchical structure, the team decided to use tags to indicate the structural role of each component (H1, H2, P).

To record the abstract structure, the team invented another document, the *Document Type Definition*, which described all the content tags and showed how those elements related to each other in the larger structure.

Using XML to mark up data will add to its size, sometimes enormously, but small file sizes aren't one of the primary goals of XML: it's only about making it easier to write software that accesses the information, by giving structure to data.

—David Hunter, *Beginning XML*

Then, in a stroke of genius, the team suggested separating format from content. In a classic mainframe approach, they moved all formatting rules into their own file. So now the team had three different documents:

- The original document, with tags indicating the role of each item in the larger structure (*this is a major heading, this is a caption*).
- A file full of formats, arranged in a series of equations. *If the tag indicates this text is a major heading, then format it in 24 points blue.*
- A file describing the abstract structure that all documents of this type must follow, laying out all the official tags in a kind of vocabulary.

IBM's General Markup Language of 1969 grew into the Standard General Markup Language (SGML) of 1974 and became an international standard in 1986. It is called a language, but really it is a machine for producing vocabularies—sets of standard tags that define the structure and content of a particular type of document. SGML had a lot of pluses:

- It aided large-scale publishing (in areas such as aircraft manufacturing and maintenance, nuclear power, telecommunications, space) because it let corporations, industries, and governmental agencies create a standard set of tags that could be inserted into ASCII documents instead of proprietary formatting codes, enabling a printer to accept thousands of documents from hundreds of different subcontractors, to blend together into a single document set, without paying for extensive translation of different formatting tags.

- It worked fine with long-lasting documents, particularly those requiring many revisions, updates, and cross references, because the structural model remained the same despite all the tinkering.
- It ensured consistency across many pages and across many books.
- It gave you a single source of raw text that could be traded back and forth among many vendors, because it was just ASCII text.

But yielding to the influence of its requesters, SGML soon acquired so many options, exceptions, and tangled syntaxes that it became too complicated for most mortals to use. Corporations had to have specialists work on a project for a few years to get the right set of tags, and then they had to convert all the old documents at great expense. Plus, the language was aimed at publishing a lot of paper. No one had heard of the Web back then. Because the SGML software ran on large computers, the code sprawled and grew heavy—not an easy applet to download over the Internet.

In the 1980s, people began to see that you could not only create text electronically, but you could deliver it that way—in e-mail, help files, and CD-ROMs. The invention of hypertext gave users a way to move around in that electronic text. Click here and go there. Taking off from these ideas, Tim Berners-Lee used SGML to come up with a Document Type Definition—a specific set of tags—for hypertext. With the Hypertext Markup Language (HTML), the Web was made possible.

While XML demands a bit more attention at the start, it returns a much larger dividend in the end. In short, HTML lets everyone do some things but XML lets some people do practically anything.

—Elizabeth Castro,
XML for the World Wide Web

HTML is just a bunch of tags. H1 means Level One Heading, and P stands for paragraph. Despite the term "language," HTML is really a vocabulary list—it is not a factory for creating new tags, like its generator, SGML.

New software, called a *browser*, downloaded the HTML page, read the tags, and, by using the Document Type Definition that Berners-Lee had created, applied its own format to each element and then displayed the result on-screen.

HTML encouraged the explosion of Web pages because its tags were:

- Flexible enough to allow you to format the text in a lot of different ways.
- Easy to write in a text editor or word processor, and (relatively) easy for human beings to read.
- Just part of regular text files (rather than proprietary binary code), making for faster transmission, low overhead, and portability from one application to another.

But when the Web took off, and individual sites grew to millions of pages, HTML revealed some limitations, such as:

- Its tags do not identify meaningful content. Yes, it is a heading, but what is it about?
- The tags can appear in almost any order because the Document Type Definition is purposely loose about structure (to encourage almost anyone to put almost anything up on the Web). But that lack of definition means that software has only a crude model of the structure of a document, and so, instead of zipping right to a structural element such as price or author, it must read every character, hoping for a clue to indicate which is which. Searching and manipulation of the contents were therefore tedious and unreliable.
- E-commerce demanded database exchanges and HTML took a lot of massaging just to get its content in and out of databases.

All that HTML tells the software is general information such as:

- This is the header part of the document.
- This is the body part of the document.
- This is a heading.
- This is a paragraph.
- This is a table.
- This is the anchor for a link; if someone clicks it, the browser should follow the path to this other page.

HTML does not tell the software that this element is a *book title*, and this one over here is an *author name*, and that both live within an element called *Book Description*. In HTML all of those elements might be tagged as paragraphs. In those circumstances, it is hard to tell software how to look within a Book Description to find the

particular paragraph that has a book title, as opposed to an author name. For ordinary consumers this failing seems trivial. But if a company wants to be able to order products and sell them over the Web, tags distinguishing one type of content from another become critical.

Also, over the years customers drove the vendors to expand the tag set to include more tags that could help format the page more attractively. But because users were so sloppy, the browser makers had to write a lot of code handling exceptions, mistakes, and problems, so the code became bloated and slow.

To resolve these problems, the eXtensible Markup Language (XML) was created in 1996, becoming a standard in 1998. XML is really just SGML lite. Like SGML, XML lets you create your own vocabularies of tags for different purposes. Because it is a subset of SGML, any old SGML processor can read XML. As it is still fairly new, only the latest Web browsers can handle XML in its pure form, so servers have to transform the XML document into an HTML one. But XML is well on its way to outshining SGML as an all-purpose markup machine, a way of generating your own tags for your own purposes.

XML tags identify the content, not the format. XML says things like:

- This document is a catalog.
- This is a book description.
- This is the book's title.
- This is the book's subtitle, if any.
- This is the book's page count.
- This is the book's price, in U.S. dollars.
- This is the book's ISBN.

Here's an example of the XML markup of the body of a marketing article. This genre is called featuresandbenefits, and it consists of a challenge followed by a solution. Over and out.

```
<featuresandbenefits>
<challenge> Today, even a small business needs a Web
site. To make it easy for customers to find out about your
products, see testimonials from other satisfied customers,
get a map to your store, you need a Web site.</challenge>
<solution>The PR Express offers a fast and easy way to
build a Web site that expresses your business
case.</solution>
</featuresandbenefits>
```

The tags are defined in a document called the Document Type Definition, which declares each element, shows what components should appear inside it, in what order, and how often. For instance, to define the tags used in the marketing article, the team declares an element called featuresandbenefits, which contains a challenge, and a solution. Then the DTD declares the challenge consists of ordinary text (grandly known as data made up of characters that will be checked by software called a parser, i.e., parsed character data). Ditto for the solution. That's it.

```
<!ELEMENT featuresandbenefits (challenge, solution)>
<!ELEMENT challenge (#PCDATA)>
<!ELEMENT solution (#PCDATA)>
```

Content tags resemble the names of records and fields, or rows and columns, in a relational database. Looked at another way, the tagged elements resemble objects in an object-oriented database. Either way, using XML tags, you have created what programmers call "structured" information (as opposed to the sloppy irregularly organized and unpredictable documents you usually produce), so a programmer can easily grab your content and send it to a database, or pull facts out of the database and plug them into your document without inadvertently putting the address where the phone number belongs. In effect, your document becomes a little database capable of talking to other databases.

In this way, XML makes it easy to write programs that manipulate the content of XML documents. Because it enforces a strict model of the content and insists that writers observe a rigorous discipline on

tagging (no exceptions, no mistakes, no craziness), XML limits the number of possible tags and organizational models so processing software like parsers and browsers can be lightweight. Any vocabulary generated in XML is methodical and precise, and therefore unambiguous (at least from the point of view of software). But, amazingly, with a little training, anyone—even you—can read an XML document. If the original tags are written well, even a newcomer can figure out what each element is, and where it belongs.

This is where the extensible in Extensible Markup Language comes from: anyone is free to mark up data in any way using the language, even if others are doing it in completely different ways.

—David Hunter, *Beginning XML*

And XML files are text files using standard codes for every character, such as the American Standard Code for Information Interchange (ASCII), or Unicode. Therefore, almost any software in the world can read these documents. Plus, the markup adds metadata, making the file "self describing" and much easier for software to digest. Total strangers, with software you never heard of, can pick up your XML document, read it, copy from it, insert data into it, and generally manhandle it. Where HTML focuses on formatting content in a browser, XML enables that, but offers software a chance to perform further processing of the contents, such as searching, sorting, reorganizing, inserting, and deleting at the behest of the user, and your own little applet downloaded along with the file.

XML is democratic, too. Anyone can create a vocabulary of tags, using XML, defining what the content elements will be, and how they will be organized, in a Document Type Definition (DTD) or schema. Once displayed on the Web in an XML document with its accompanying DTD, those tags and structures are open to anyone else to read and manipulate. They are not proprietary or hidden, so other people can disassemble the document, reuse parts of it, and translate the tags into their own vocabulary. Industries can adopt standard vocabularies. Businesses can have their own variations and easily translate content back and forth.

Fundamentally, XML separates the structural model, the actual content, and the rules for formatting the content.

- The Document Type Definition defines the abstract structure for this type of document, creating a vocabulary and organization of tags.
- The document itself uses those tags to label the actual content.

- One or more stylesheets tell the browser what format to apply to the content, using those tags, in different circumstances (old browser, new browser, handheld device).

That division of labor often means that you don't get to see exactly what your text will look like—a step back to the awful days of UNIX editors like vi, or worse. But you can have your software display the text in a way that imitates one or another of the stylesheets, while an inset window displays the standard structure, and the software beeps at you if you get out of line and try to insert the wrong element next, or delete a required element. You are still writing, but at your elbow is a structure monitor. You have lost the independence of word processing, where you can organize and format any way you like. But your text is becoming a lot more useful for your users.

You are leaving the free-wheeling world of documents and entering the world of carefully defined, standardized objects—steps, captions, definitions, introductions, whatnot. Each element that has been identified with an XML tag now acts as a discrete object. Often, it is stored as an object in an object-oriented database, but most writers do not care how the software stores the element. What does matter is that now, instead of creating whole documents and formatting them, you have to craft a set of distinct objects without always knowing in advance all the structures into which they may be fitted together and formatted.

See: Coombs, Renear, and DeRose (1987), Goldfarb (1990), Goldfarb and Prescod (2000), Hunter et al (2000) , Kay (2000), McGrath (1998), Megginson (1998), Musciano and Kennedy (1997), Travis and Waldt (1995), Turner, Douglas and Turner (1996), Young (2000), Yourdon (1994).

Building Informative Objects

Think of an object as a little creature that has only one purpose in life: to apply certain methods to a specified collection of data.

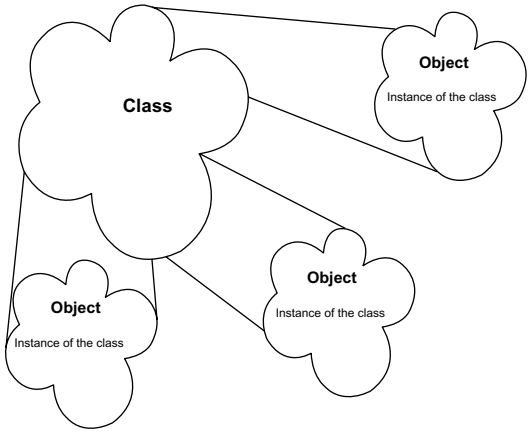
—James Martin

The biggest change for writers moving into the world of XML is that now you must think more rigorously about structure, creating little Lego bricks of content, destined to be assembled and reassembled in ways you cannot always anticipate. You are no longer making a single document; you are preparing objects that can be put together in many different structures, formats, and media. But what is an object?

Programmers have their own ideas of objects, described in pattern languages and documentation. But as a writer you want to create chunks of text to communicate: you are making informative objects.

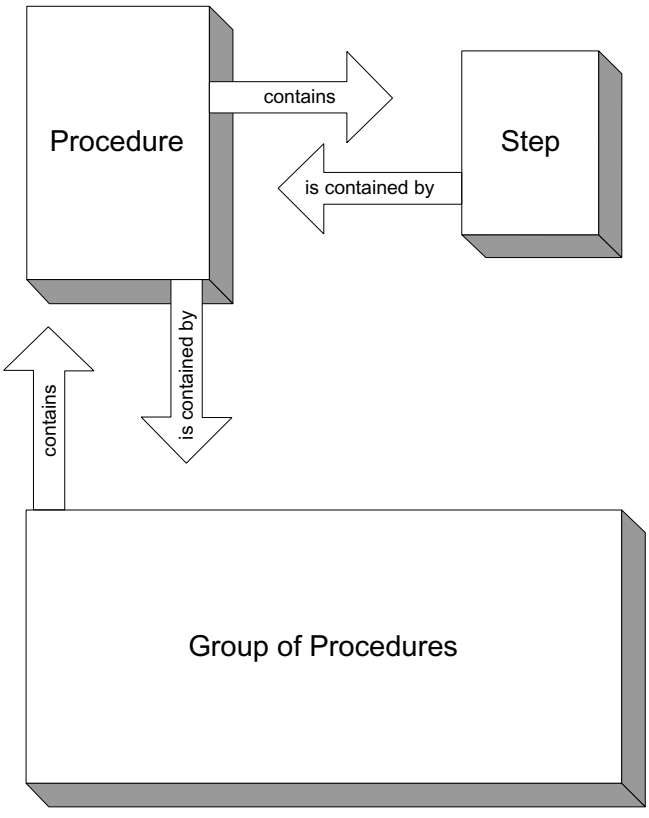
From a writer's point of view, objects have seven characteristics that make them informative. We'll cover those here. If you are familiar with object-oriented programming, you'll recognize the analogies here. But even if you don't live in an object-oriented environment, you'll begin to see some of the benefits—and the challenge—of organizing your content in objects.

1. Each object starts life as a category of content.



Each object is defined abstractly as a class, such as a Procedure. The abstract object is an ideal notion, like a Platonic form—a category of information. As soon as you write something, you are creating an instance of that class—a unique example of the class, a particular real object.

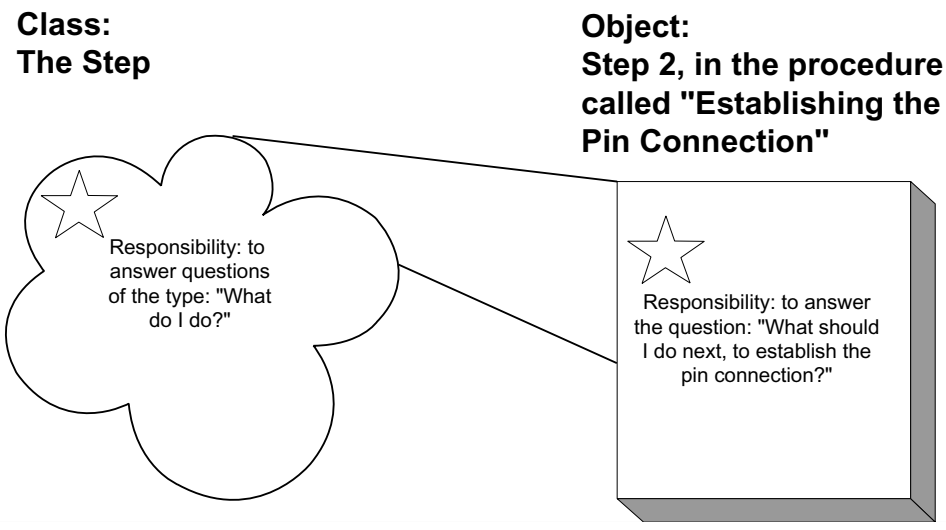
2. The class has a standard structure.



The class defines its components in a certain sequence within a single hierarchy. The standard tells us how many components are allowed or required, in what order, and which ones are just optional. The relationships between objects are defined in a content model, known in XML as a Document Type Definition, or schema.

Every instance of that object must follow the same internal pattern.

3. Each type of object has a job to do.



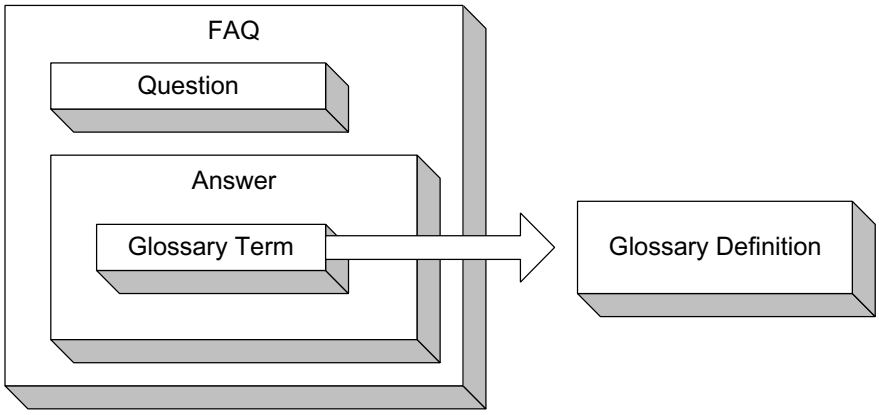
The responsibility or function of each informative object is, in its simplest form, to respond to a particular type of question, such as, "What are the results I can expect if I do this step?" (Answer: an object called an Explanation.)

Over the centuries, writers in any genre have invented common elements to answer the kind of questions that keep coming up. For instance, in creating procedures, writers came up with a standard element to answer the question, "What do I do next?" That element, the step, shows up in so many manuals that we can now say it is conventional. Turning a routine element like that into an object is easy. We recognize the element from our reading, and we are simply blessing it with a new description and a new role. Informative objects are often familiar elements doing a traditional job in new clothes.

An individual object does not make a system. A system is made up of a number of objects. A model of a commercial organization usually consists of many objects that are related in some way.

—Ivar Jacobson, Maria Ericsson, and Agneta Jacobson,
The Object Advantage

4. Objects talk to each other.



Being electronic, and living within a world of software, each object can exchange messages with other objects. Yes, these objects can link up. Basically, one informative object sends a message to another saying, "Will you display yourself?" If the linkage works, the other object shows up on-screen. Unlike the complexity of messaging between heavy-duty programming objects, this little exchange is essential to our job as creators of interactive text.

5. The same object can be reused in many different locations and media.

Each object can be used as a component in other objects, placed in a different position in a sequence of objects, or used without some of its optional components.

Re-use goes more easily with objects because:

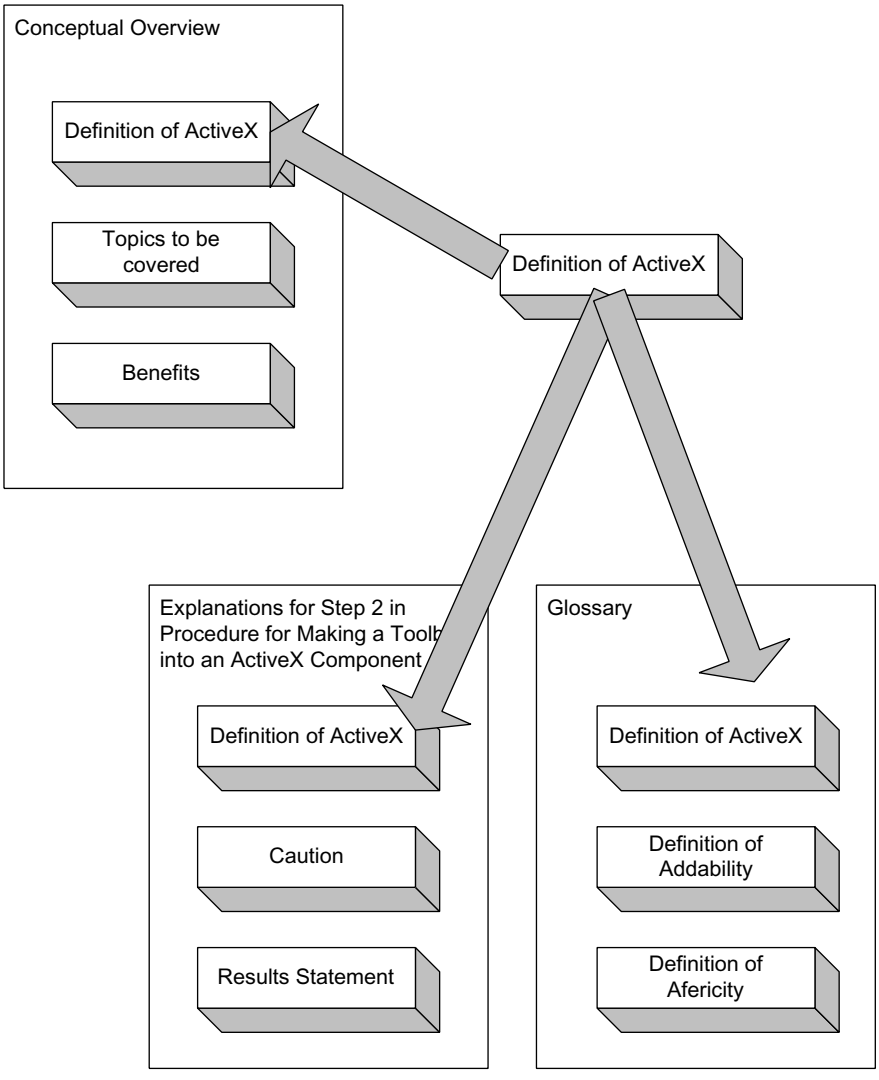
- We can use exactly the same object in many locations without rewriting it. (This kind of reuse can save hours of our time, and thousands of dollars in the budget.)
- We can select certain objects to be sent to a handheld device, others to go to a Web page, and still others to be printed out on paper simply by picking a different Document Type Definition or stylesheet without any rewriting.
- When updating material, we can search for a set of objects by class name. (Give me all the steps in this long procedure.)

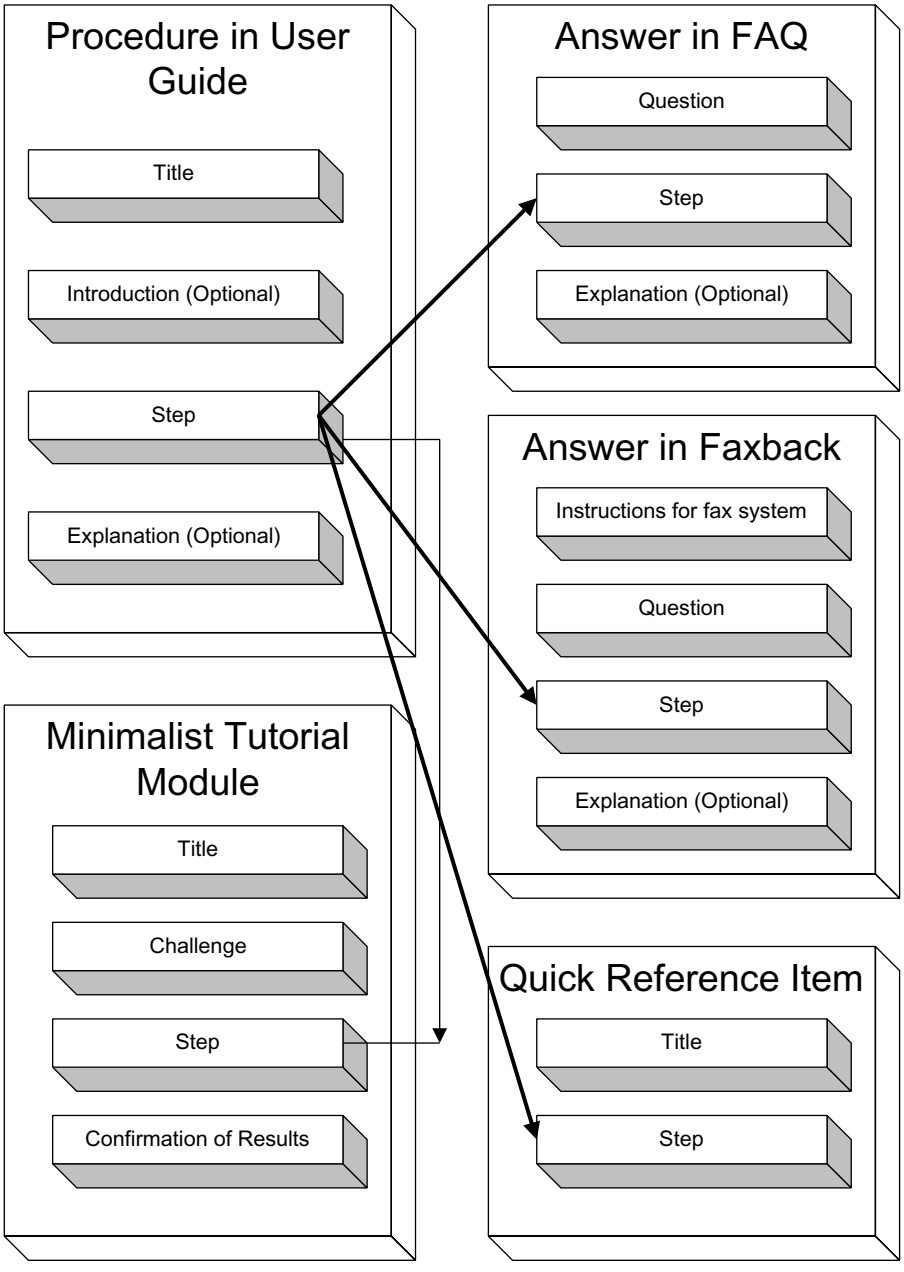
Nothing can have value without being an object of utility. It it be useless, the labor contained in it is useless, cannot be reckoned as labor, and cannot therefore create value.

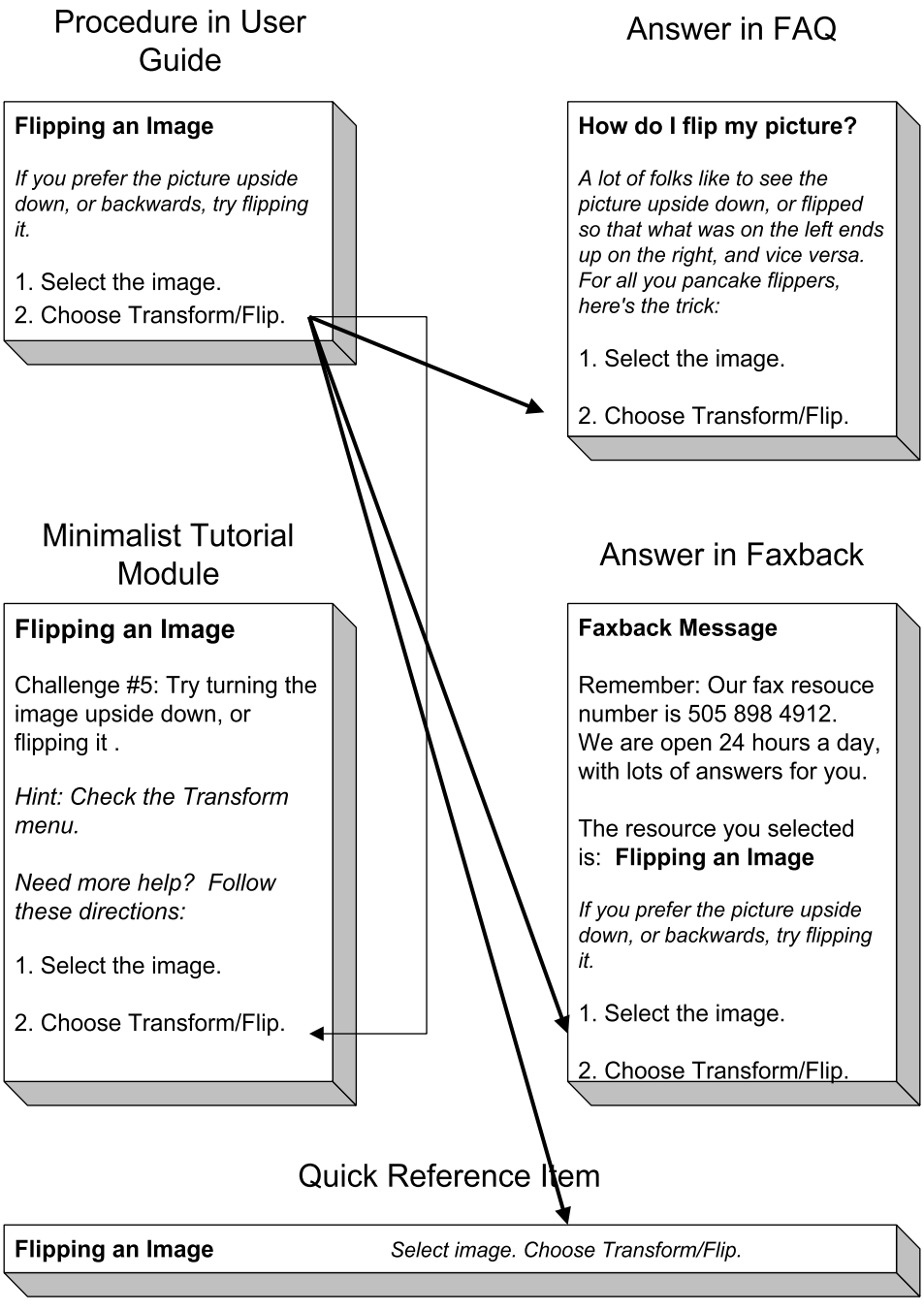
—Karl Marx, *Capital*

That way we can make a change in the steps without having to paw through the intervening explanations.

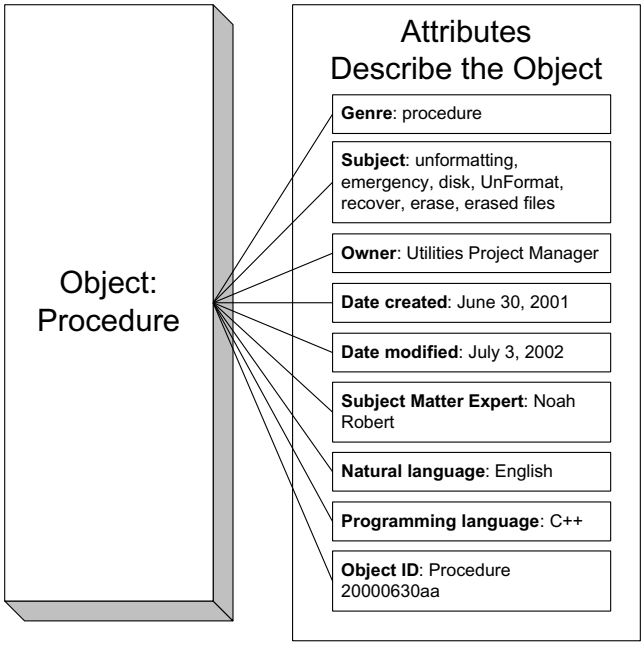
- We can take apart an existing object and use selected components, but not others, in a new object.




Procedure in User Guide
Title
Introduction (Optional)
Step
Explanation (Optional)
Answer in FAQ
Question
Step
Explanation (Optional)
Answer in Faxback
Instructions for fax system
Question
Step
Explanation (Optional)
Minimalist Tutorial Module
Title
Challenge
Step
Confirmation of Results
Quick Reference Item
Title
Step


Procedure in User Guide
Flipping an Image
If you prefer the picture upside down, or backwards, try flipping it.
1. Select the image.
2. Choose Transform/Flip.
Answer in FAQ
How do I flip my picture?
A lot of folks like to see the picture upside down, or flipped so that what was on the left ends up on the right, and vice versa. For all you pancake flippers, here's the trick:
1. Select the image.
2. Choose Transform/Flip.
Minimalist Tutorial Module
Flipping an Image
Challenge #5: Try turning the image upside down, or flipping it .
Hint: Check the Transform menu.
Need more help? Follow these directions:
1. Select the image.
2. Choose Transform/Flip.
Answer in Faxback
Faxback Message
Remember: Our fax resouce number is 505 898 4912.
We are open 24 hours a day, with lots of answers for you.
The resource you selected is: Flipping an Image
If you prefer the picture upside down, or backwards, try flipping it.
1. Select the image.
2. Choose Transform/Flip.
Quick Reference Item
Flipping an Image
Select image. Choose Transform/Flip.

6. Searches can turn up individual objects, thanks to attributes.

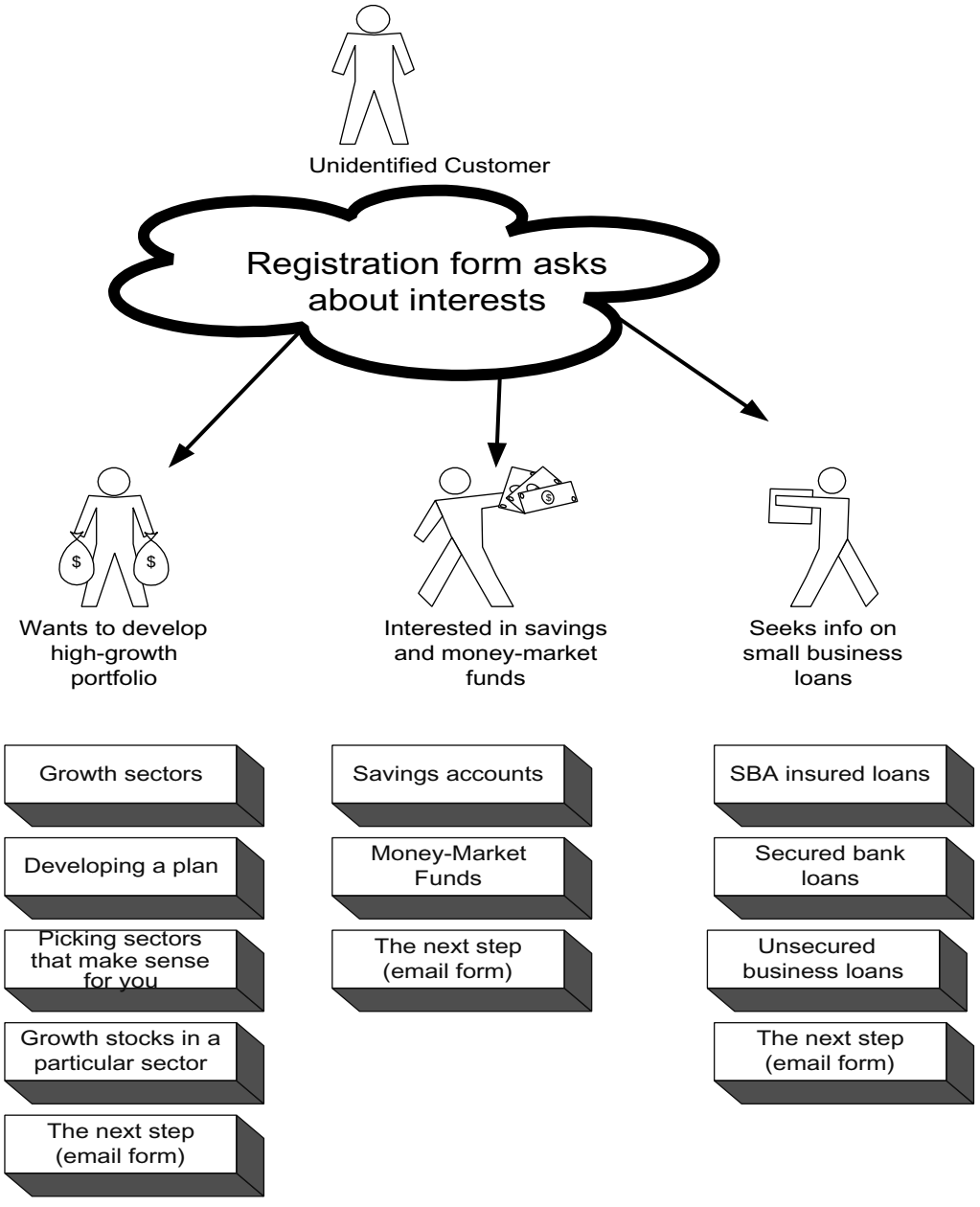


Attributes describe the object. For instance, the Procedure object has attributes such as subject and owner; when you create a new procedure, you fill in the blanks with the new subject and the person who "owns" that particular object.

Each class of objects has a set of attributes, like fields on a record in a database, and each instance of that class has its own values in those attributes. The point is to allow users (and writers) to search on values in a variety of attribute fields, such as:

- Genre
- Subject
- Owner of the information
- Date created or modified
- Subject Matter Experts or Authors
- Products named
- Product ID
- Natural language used in the writing
- Operating systems this product runs on

7. Objects can be assembled quickly, creating personalized content.



Personalization demands that you break up your content into small chunks so you can deliver just the pieces one person wants without the annoying extra details they would like to avoid. Customer relationship management, Web application server, customization and personalization programs can track individual visitors by their cookies and offer personal welcome pages with customized content, including topics the person has indicated interest in—but only if you have carved your material up into a lot of bite-sized chunks.

A use case is a sequence of transactions in a system, initiated by a user of the system. A use case has a complete flow of events, that is, it has a well-defined beginning and an equally well-defined termination.

—Ivar Jacobson, Maria Ericsson, and Agneta Jacobson, ***The Object Advantage***

Taking an object-oriented approach to structure cuts through the noise

Looking at the elements in current documents as informative objects can help you improve the effectiveness of your writing. Having an object model to work with (taken from the DTD), you can do a number of things that tend to make your writing more alert. You can, for instance:

- Define the purpose of each object you use so you can include only what is relevant, and leave out the interesting but extraneous details that confuse people.
- Follow the standard pattern for these objects so readers (and programs) know what to expect and do not get derailed by strange variations.
- Apply that pattern in editing old documents to spot places where the document is not working properly, correcting those passages without getting lost in stylistic revisions.
- Create team templates that restrict and direct new writing.
- Build in attributes that allow more pinpointed searching for particular small objects, such as a definition or a single procedure. You are making it possible for software to filter out thousands of near misses and duds, focusing on the particular type of object you or your users want.
- Make your conceptual model explicit and articulate. Exposing your structure generally makes for faster reading and better comprehension.
- Support the users as they grasp and use the structural model you present by being consistent in your own use of

it, avoiding internal inconsistencies that raise questions in people's minds.
- Offer multiple perspectives on the same information and various organizations on different media.
- Identify possible audiences for particular objects using an attribute such as Audience.

The Web drives your organization to take traditional types of documents, such as the data sheet or procedure, and standardize them, defining each one as a distinct genre with a clearly articulated purpose, audience, subject, and organization. At first, moving into this environment feels constricting, artificial, and a bit abstract. But after a while you'll probably find that your writing goes faster, gets cleaner, and does its job more efficiently because you know what the purpose of each object is, understand what kind of question it answers, and you do not get distracted, drawn off on a detour, or just plain confused about what goes where. You begin to think structurally, and, interestingly, that often leads to uncovering new facts, understanding your subject in more depth, and shaping your material for the benefit of your readers, not yourself, or your teammates. What you give up in originality you gain in impact.

Business rules go hand in hand with customer profiles. By combining the two, you can begin to target the right information, products, and offers to your customers.

—Patricia Seybold and Ronni Marshak, *Customer.com*

See: Ames (2000), Belew (2000), Bradley (2000), Coad and Yourdon (1991), Coombs, Renear, and DeRose (1987), Gamma, Helm, Johnson and Vlissides (1995), Goldberg and Rubin (1995), Goldfarb (1990), Goldfarb and Prescod (2000), Herwijnen (1990), Hunter et al (2000), Jacobson, Ericsson, and Jacobson (1994), Kay (2000), Khoshafian and Abnous (1995), Lee (1993), Lie and Bos (1999), McGrath (1998), Megginson (1998), Object Management Group (1997), Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen (1991), Taylor (1992), Travis and Waldt (1995), Turner, Douglas and Turner (1996), Young (2000), Yourdon (1994).

POST

My Idea:

Post to HotText@yahoogroups.com

Express your own idea on:

HotText@yahoogroups.com

Subscribe:

HotText-subscribe@yahoogroups.com

Visit:

http://www.WebWritingThatWorks.com