stc.org/region8/nbc/www/

northbay news

The monthly newsletter of the NorthBay Chapter of the Society for Technical Communication

Volume 7, Number 8, August 2000

"Oops, Here comes OOP."

In This Issue

society for technical communication

Writing for Programmers, Part 3

Ken Delpit, Newsletter Staff

This is the third of a three-part series of articles about writing for programmers. The series began following a presentation ("Programming Concepts and Terminology for Technical Communicators") by Andrea Ames, STC Region 8 Director-Sponsor, at the March chapter meeting.

In Part 1 (April 2000), Andrea assured us that technical writers have what it takes to write about object-oriented programming (OOP) without having actual programming experience. In Part 2 (May 2000), we explored the fundamental concepts and meaning of OOP, and compared it to traditional programming. In Part 3, we discuss the OOP application development process, compare various OOP environments, and consider the role of the OOP technical writer.

How Are OOP Applications Developed?

" The basic building block out of which all living things are composed is the cell. Cells are organic 'packages' that, like objects, combine related information and behavior." *–David A. Taylor*, in *Object Technology, A Manager's Guide*

Continues on page 4 ☞

Consistency and Competition

John Dibs

This issue features the finale of Ken Delpit's three-part series, *Writing for Programmers*. The quality and length of this series reveals Ken's and Andrea's **consistent** effort to tackle a difficult and an important topic. The writeup of Paul Kaldunski's presentation, "Bandwidth to Burn," is forthcoming in the next issue.

Competition will be a topic for our upcoming open meeting on August 17. Sponsored by Bay Area STC chapters, Touchstone is a local competition for technical communicators. Distinguished entries from Touchstone are entered in the STC International Technical Communications Competition. Due to the shear size and location of our Bay Area chapters, expect several entries from our region to reach this prestigious accomplishment level.

submitting articles and ads
We welcome articles, advertising, and news about meetings, workshops, and courses that pertain to technical communication. Please email simple text to the editor at jdibs@earthlink.net
For our current advertising rates, please email or phone the editor.

reprints and distribution
If you reprint articles from the *northbay news*, please credit them and forward a copy to the editor. Reprints in non-STC publications are subject to the author's approval. Copyright © 2000 *northbay news*. *northbay news* is free to NorthBay Chapter members. Nonmember subscriptions are $6 per year.

STC Mission Statement
The mission of the Society for Technical Communication is to improve the quality and effectiveness of technical communication for audiences worldwide.

This Month's Meeting

Thursday, August 17, 2000

• Northern California Technical Communications Competition (NCTCC) Winning Entries
• Open Forum and Discussion

Join us this month as Whitney Parker reviews the NCTCC winning entries and we have an open discussion on topics of interest.

- View the winning entries for the 1999 NCTCC
- Learn about the various documentation categories and opportunities for entering a discussion and judging in this prestigious STC competition
- Join in discussion about topics of interest to technical communicators

Short presentations for the open forum portion of the meeting are welcome. Please contact huget@hooked.net in advance of the meeting if you are interested in giving a mini presentation.

Please bring your ideas and questions to this month's meeting!
Cost: $2.00 members; $4.00 non-members

Meeting Time & Schedule

Date:	Thursday, August 17, 2000	
Location::	Parker Compumotor, 5500 Labath Dr., Rohnert Park	
Time:	5:30 - 6:30	Networking and Refreshments
	6:30 - 8:15	Introductions and Program
	8:15 - 8:30	More Conversation, Idea Swapping

Welcome new STC members!

Francesca Flynn
Liz Kaiser
Christopher Myrick
Will Ross

Welcome STC members transferring to the NorthBay chapter!
Rolfe Dlugy-Hegwer
Andrew Dugas
Janice Fitzpatrick

The Pan-Pacific Conference

Registration for the Region 7/8 Pan-Pacific Conference is running at a record clip, but there is still time to register. Said conference chair Jack Molisani, "We're delighted at the strong turnout to date. Attending this conference means an opportunity to network with fellow professionals, an opportunity to learn, and of course an opportunity to enjoy getting to know some great people!" The conference takes place October 19–21 in Honolulu, Hawaii.

Molisani continued, "There's still space for more registrants." While rooms at the conference hotel, the Renaissance Ilikai on Waikiki Beach, are nearly booked (and may be sold out by the time this article is printed), rooms of comparable quality and price have been reserved just steps away from the Ilikai, at the Hawaii Prince. Furthermore, with some 100 workshops and sessions planned for the conference program, there will be plenty of sessions to accommodate everyone at the conference.

Surprisingly Affordable

Conference costs are surprisingly low, and "are a bargain considering the incredible value of this conference," said Molisani. The conference fee is $275, and hotel accommodations can be had for as little as $57.50 per person, double occupancy. Discount airfare is also available.

For best travel rates, contact the conference travel agency, Seawind Tours and Travel, at 1-800-424-3324, or email stcppc@seawindtours.com.

To register for the conference, download the preliminary program from the conference Web site: www.pan-pacific.org.

Conference Program: Abundant Choices

Every communicator will leave this conference with much more than just a suntan. Program sessions address the needs of beginners and experts alike. More than 100 speakers will share their knowledge and insights with conference participants in 70 sessions and 32 full- and half-day workshops. Topics cover writing and project management, career development, knowledge management, mentoring, collaboration, contracting and job searching, project management, distance learning, training and development, and even contract law and ethics. In addition, as befits an international conference, translation and writing for international audiences is thoroughly covered.

The keynote speaker for the conference is Martha Baer, senior contributing editor at *Wired* magazine. Baer has served as a writer, editor, consultant, producer, and research chief for the publication, guiding it to its edgy, pre-eminent voice while covering technology and Silicon Valley. Her wide-ranging talk will deal with technical style, audience, and interface issues, as well as the state of our profession as seen from one of today's most exciting publications.

Tours and Activities

If you find the idea of spending time on the most beautiful beach in the world appealing, this conference is for you. And what's a trip to Hawaii without a luau? You'll have an opportunity to sample the fare of the islands and the culture of Polynesia at our luau on Thursday night, as well as take a dinner cruise on Saturday to view the magnificent Waikiki waterfront!

For More Information

Visit the conference Web site, www.pan-pacific.org, for more information about the program. For travel information contact Seawind Tours and Travel at 800-424-3324 or 800-949-4144. For any other question, contact conference chair Jack Molisani, jackm@ClarityTechnical.com.

About the author: Jay Mead is past-president of the STC Rocky Mountain Chapter, and PR chair for the Pan-Pacific Conference.

TheNCTCC Committee Needs You Now!

What

Volunteers to provide expertise in the following areas for the STC Northern California Technical Communications Competition (NCTCC) Committee this year.

Who

- NorthBay Chapter Liaison to attend committee meetings and serve as a communication conduit for our chapter.
- Graphics Artist for design work on this year's NCTCC Competition materials — shape the image for this highly visible STC effort.
- Web Weaver for content and user interface design, maintain web page throughout competition, and enhance as appropriate (may be same person as Web Page Techie)
- Web Page Techie — May be same person as Web Weaver to assure smooth functioning of Competition Web page — technical effort will include arranging for hosting, linking, and testing of page(s) — will include on-line forms for submitting entries, applications to serve as judge, judging forms, etc.

Why

Participating in this highly visible, respected competition on the NCTCC Core Committee gives you the chance to demonstrate your talents to peers and companies doing technical communications work throughout Northern California, hone skills, add to your resume, and enjoy the work with a great group of volunteers.

When

Immediately if not sooner. Kickoff planned early in August.

How

Contact Judith Herr, NCTCC Chair, herrj@home.com for information, or to volunteer (925-443-4515). For the chapter liaison volunteer role, please contact John Dibs (707-792-1791)

Writing for Programmers

Continued from page 1

Andrea describes the application development process as a series of five basic steps:
1. Design
2. Code
3. Compile
4. Test and debug
5. Deploy

Ideally, all products will have progressed through each step before they are deployed, and all products will remain in a step until the product has been sufficiently "cooked." Only when the product has met all criteria will it be promoted to the next step. Ah, but the ideal is an elusive goal in software development.

Software development is not like fast food: "sufficient cooking" is required.

For a couple of reasons, these steps are not strictly linear. For one, software development is, by nature, an iterative process. Systems are designed, built, and tested. Testing invariably reveals problems with the implementation, and sometimes with the design, so the cycle is repeated until sufficient performance and reliability are achieved, or until the budget runs out.

For another, steps are sometimes shortened or skipped entirely, and not always for the best reasons. Many computer users have experienced a program that functioned so erratically, it made them wonder how the product ever made it out the vendor's door. Steps 2 and 3 are essential for reaching step 5, Andrea says. After all, you cannot possibly deploy a product unless you have, at least, coded and compiled it. Unfortunately, she says, steps 1 and 4 are too often considered optional in the application development process.

Let's look at each step in more detail.

Step 1: Design

"To err is human, but to really foul things up requires a computer." — *Anonymous*

In designing an object-oriented application, one must first identify the objects. Design doesn't stop there, however. Andrea outlines five stages of OOP object design:

- Object discovery. Determine which object types will provide the functions of the application and solve problems for the user.
- Object assembly. Build the objects and discover the data (properties) and functions (methods) that team members must provide for the objects to work properly.
- System construction. Bring the objects together in a final application.
- System extension. Discover how well, or how poorly, the application is designed according to its *extensibility*. If it is not easy to extend, find the weak points and fix them.
- Object reuse. Build new systems by reusing objects in other applications or subsystems. This process tends to identify those objects that must be changed significantly in order to fit a new, similar purpose. When an object *should* be reusable, but isn't, chances are that it should be modified for its original purpose as well.

Notice that these design tasks go beyond the nominal "design" phase (step 1). In fact, good system design spans the entire application development process. It is a paradoxical reality of most software development, including OOP, that the design isn't really final until the development is complete.

Inevitably, as developers progress in a project, they discover problems, nuances, and improvements that affect the design. Depending on real-world considerations, such as software quality and extensibility requirements, as well as project budget and urgency, modifications to the original design are often required well into the project.

Proactive companies can take measures early in the design phase that help minimize subsequent design changes. Depending on the resources and time available, companies can employ GUI mockups and product prototypes to gauge the look and feel before the programmers begin development in earnest.

Even better, companies can make sure the application will accomplish what their potential customers want. Prerelease demonstrations and mockups can be presented to test users and focus groups for feedback. Companies can engage in market surveys to assess customer needs, and they can commission technical analyses to measure the feasibility and potential pitfalls of the application. Writers, as advocates for the reader (user), are in a terrific position to help with many of these measures!

Step 2: Code

"The sad thing about artificial intelligence is that it lacks artifice and therefore intelligence." – *Jean Baudrillard (b. 1929), French semiologist*

Programming, or *coding*, is the process of writing *source* statements in the chosen programming language so as to turn the design into a working product. Modern programming languages are *high-level* languages. That is, a single source statement usually results in several *machine-language* statements. Freed from having to write many tedious, low-level machine-language statements, programmers' productivity is effectively amplified.

Continues ☞

Writing for Programmers

Continued from page 1

However, computers cannot execute high-level source statements directly. A computer's native language, the only language that it can "speak," is the one that is burned into its silicon. Therefore, all source programs must be converted into machine language before they can be run, or *executed*. There are two basic types of source-to-machine-language translation processes: *compilation* and *interpretation*.

Run-time interpretation presents a key benefit for the user as well as for the developer... With a separate interpreter for each platform, programmers can develop a single source application, and deploy it on multiple platforms!

Many high-level languages, each with its own inherent strengths and weaknesses, are available for the programmer's toolbox. "Legacy" procedural languages, such as Pascal, COBOL, and FORTRAN, are still being used, though their star is fading. Object-based languages, such as VisualBasic, occupy a middle ground between legacy languages and true object-oriented languages, such as C++, Java, and Smalltalk.

Step 3: Compile

"You have riches and freedom here but I feel no sense of faith or direction. You have so many computers, why don't you use them in the search for love?" – *Lech Walesa (b. 1943), Polish trade union leader, politician*

Both *compilers* and *interpreters* read a program's source statements and translate them into machine language. Compilers and interpreters differ in *how* and when they do the translation. Both have their advantages and disadvantages, and both have their places in today's programming landscape.

Compilers are run, usually in a batch process, during application development. Compiler output is stored, so that once compiled, programs can be run as many times as desired without having to be compiled again.

Interpreters and scripting languages, on the other hand, are called into service each time a user wants to run the application. Generally, interpreter output is not stored permanently, so that interpreted programs must be interpreted each time they are run.

Andrea illustrated the distinction between compilers and interpreters with a culinary analogy. Suppose a spouse brings back to America a recipe for spaghetti sauce from his ancestral Northern Italy. The recipe is written in Italian, and the "computer" (that is, the cook) speaks only English. So someone must sit down with an Italian-English dictionary and a calculator and translate the words and metric units into English equivalents.

An interpreter would do the translation each time the recipe is used. A compiler would do the translation once, and would save the translated recipe for future use. The compiled recipe would be "executed" faster, but would not recognize recent changes to the original Italian source recipe, as would the interpreted recipe.

Compared to interpreted programs, compiled programs are small and fast. Today, most conventional applications, such as word processors and spreadsheets, are compiled. Given that a computer can execute only machine-language statements, separate versions of compiled applications must be created for each platform on which they are to be run. And compiled programs are inflexible—they must be recompiled before any changes to the source language can take effect.

While comparatively slow in execution, interpreted applications offer distinct advantages over compilers. For one, the source code itself can be changed right up to the time it is run. This flexibility is a time-saver for developers, and also allows the possibility of dynamically tailored programs. Moreover, run-time interpretation presents a key benefit for the user as well as for the developer—the opportunity for platform independence. With a separate interpreter for each platform, programmers can develop a single source application, and deploy it on multiple platforms!

Step 4: Test and debug

"Perfection of means and confusion of goals seem— in my opinion— to characterize our age." – *Albert Einstein (1879–1955)*

One would hope that all software is thoroughly and rigorously tested before it is marketed. And one would be naïve to think that this happens routinely. All too often, it seems, the desire to get to market first, or at least early, cause marketing objectives to displace technological

Continues ☞

Writing for Programmers

Continued from page 1

excellence as the prime objective.

Some companies do an admirable job of balancing product timeliness and quality. Others cut the testing phase short, or skip it entirely. Granted, no software product is ever perfect or completely bug-free, but too many products are deployed before they're ready. Effectively, companies that skimp on testing are letting their users perform the testing.

When done conscientiously and systematically, usability testing can be a company's best friend, not to mention the users' inside ally. Usability testing can help a product hit the intended target, reveal embarrassing or critical flaws before they surface in the real world, improve ease of use, and identify potential enhancements for future versions.

Unfortunately, as confirmed by Mic Vandersluis of Parker Compumotor at the June meeting ("A Case Study in Usability"), it takes a while for companies to see the light when it comes to usability testing. Naïve project managers regard usability testing as a simple completion task that they can tack on the end of a project, like a final layer of varnish. Enlightened project managers recognize usability testing as an ongoing, essential part of product development and as a tremendous contributor to customer satisfaction.

Step 5: Deploy

"Seventy percent of success in life is showing up." — *Woody Allen (b. 1935)*

The somewhat militaristic term "deployment" is in vogue at many businesses. Referring generally to the act of delivering a product to customers, the word can take many forms. It can represent the completion of a long, arduous journey through the development of a major new product, or it can represent a simple product update. It can be accompanied by gala fanfare, hired entertainers, and unrestrained marketing giddiness, or it can be announced by a mere postcard or e-mail.

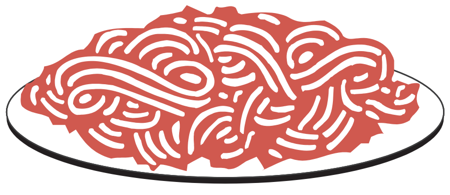
Whatever form it takes, deployment usually means the nominal end of the line for a product version. Whether it also means the beginning of a marketing disaster or a technical support nightmare depends largely on how well steps 1 through 4 were performed.

How Do Various Object-Oriented Languages Compare?

"Man is still the most extraordinary computer of all." — *John F. Kennedy (1917–1963)*

C++ is a compiler, producing generally compact, quick programs, offering true OOP capabilities. It is somewhat portable in that C++ compilers exist on nearly all platforms. Still, because of operating system-specific functions and differences between compiler implementations, significant portions of C++ programs must be converted manually before the programs will execute on another platform.

Java is neither a compiler nor an interpreter exclusively, but is sort of both. The Java compiler produces an intermediate form of code, *bytecode*, that lies somewhere between source and machine language. The Java Virtual Machine (JVM), a run-time interpreter, then interprets the bytecode rather than the original source code. While not a true compiler, this gives Java some of the advantages of a compiled language, mainly faster execution speed, while retaining its platform independence. JVMs for multiple platforms are free from various sources, such as Sun Microsystems, the original developer of Java.



Created by an English-speaking cook from an Italian recipe with French ingredients in Greek cooking pots...oh my, it's OOP.

Though it hasn't taken off like Java, Smalltalk is also a true OOP language. Developed at Xerox Palo Alto Research Center (PARC) around 1980, Smalltalk was a pioneering effort at creating object-oriented environments and graphical user interfaces. Many products today, including Windows and Macintosh operating systems, as well as Java, owe much to Smalltalk's trailblazing efforts. Nonetheless, Smalltalk's time has probably come and gone. Smalltalk is, as Andrea says, "not widely used for any practical purpose."

Java programs can be run in two ways, as *applications* or as *applets*. Java applications are run "standalone," outside of a Web browser. Java applets can be run only in a Web browser environment, and consequently must follow the security policies of both the browser and of Java. Both applets and applications may use any and all features of the Java language.

Security is a key benefit of applets. Security is integral to Java's core and thus cannot easily be compromised. Neither the applet user nor the applet programmer, for example, can "touch" a computer's hard disk directly. They cannot delete files, check license registrations, or do anything not specifically authorized by Java security.

In contrast, ActiveX components provide dangerously loose access into your computer for unscrupulous and bumbling programmers. A part of Microsoft's proprietary component object model (COM) architecture, ActiveX is ". . . specifically designed to facilitate distribution of components over high-latency networks and to provide integration of controls into Web browsers." (Excerpt from Microsoft Web site)

Continues ☞

Writing for Programmers

Continued from page 1

ActiveX browser components are run as applications. In short, nothing truly reliable controls them and users are forced either to provide their own security or to do without. To protect themselves, ActiveX users should set options in their browsers to issue alerts when potentially suspicious events, such as imminent downloads, occur. Besides the burden of setting up adequate security manually, these precautions can be an even larger pain during a typical interactive Web session.

Despite the similarity of names, JavaScript is not directly related to Java. JavaScript is a scripting language developed by Netscape Communications. JavaScript, however, is not a true object-oriented language, and its performance suffers when compared to Java because JavaScript is not compiled into intermediate bytecode.

Basic online applications and functions can be added to Web pages with JavaScript, but the number of available API functions is far fewer than the number available with Java. JavaScript code is generally considered easier to write than Java, especially for novice programmers. A JavaScript-compliant Web browser, such as Netscape Navigator, is required to run JavaScript programs.

You don't read the code. You pattern match it...So, you don't have to be a programmer to write about programming.

The Unified Modeling Language (UML) is a set of an object-oriented analysis and design (OOA&D) methods intended to help developers design object-oriented applications. Based on OOA&D methods of Grady Booch, James Rumbaugh, and Ivar Jacobson, UML is now a standard of the Object Management Group (OMG). UML provides a system for envisioning and organizing the objects in an application—before coding begins.

Perl (Practical Extraction and Report Language) is an interpreted language, based on C and several UNIX utilities. A program in Perl is known as a script. Perl has powerful string-handling features for extracting information from text files, and is often used for system administration tasks.

PERL was devised by Larry Wall at NASA's Jet Propulsion Laboratory. Lately, the popularity of Perl has exploded because of the language's power to provide interactivity in a Web page and to manipulate Web content. Though not originally designed as an OOP language, an object-oriented version of the language (Perl 5) has been introduced.

What Is the OOP Writer's Role?

"You don't read the code. You pattern match it." — *Andrea Ames*

So, you don't have to be a programmer to write about programming. As Andrea advises in the quotation above, simply knowing some programming fundamentals and being able to recognize patterns in source code give you the tools you need to provide useful documentation for programmers. "If your job includes writing code samples, then you might want to take a programming class," she says. "Otherwise, you can talk about the code without having written it," she says, distinguishing between concepts and syntax. Andrea recommends the "algae-skimmer version of programming concepts" for your OOP education. That is, you should skim just the broadest, most abstract concepts, and move on.

It helps to understand the programmers' integrated development environment (IDE) and their application development tools. Programmers use the IDE to enter, compile, test, and debug their programs. By knowing the IDE and understanding the purpose of the application, you can, for example, set up testing scenarios for programmers to use.

If you spend any time at all writing for programmers, chances are you will be asked to develop API (application programming interface) documentation. The API is "the façade," Andrea says, the face it presents to the outside world. The API provides other programmers with the information they need to develop complementary programs and subsystems, so API documentation can be critical to a product's success.

If you find yourself documenting APIs for a Java application, you may find JavaDoc particularly useful. JavaDoc is a tool for generating API documentation in HTML format from formatted comment (nonexecutable) statements inserted into the source code. The comment statements may be inserted either by a programmer or by a technical writer. It may seem daunting at first to be inserting statements into programmers' source code, but you won't affect the program's execution as long as you follow the simple comment syntax rules.

JavaDoc recognizes both its own comment statements and basic elements of the program, then produces documentation automatically. Sounds easy and useful, right? Yes, but there are a few caveats: (1) Don't expect publication-ready documentation without intervention. In all likelihood, you will want to revise JavaDoc's output for format and content. (2) Don't expect to build a

Continues ☞

Writing for Programmers

Continued from page 7

document incrementally from different runs of JavaDoc. JavaDoc generates one and only one document each time you run it. It cannot modify or directly incorporate results from previous runs. (3) Don't expect to run JavaDoc with source code from other languages, such as C++ or VisualBasic. JavaDoc runs only with Java source code.

Whatever type of programming documentation you develop, you will be doing your programming readers a service if you model the documentation after the software itself. That is, you should strive to make the documentation familiar to the programmer in structure, format, and terminology. For example, in a chapter that describes a particular class, you can identify objects within that class with level-one headings and the objects' properties and methods with level-two headings. "The best API documentation I've seen follows the code," says Andrea.

Programmers use *object models* to represent their OOP projects. An object model depicts—usually with a combination of diagrams and text—the hierarchy and elements of an application. An object model is an ideal way to present API documentation to a programmer. If the object model doesn't exist outside a programmer's head, then you can offer to help create one—a great opportunity to learn the product and build your document's structure at the same time.

If you are developing online documentation, such as HTML help or JavaHelp, an object model diagram makes an ideal "home page" or table of contents for your document. By clicking a graphical representation of an object, the reader can display properties and methods for that object. In like fashion, the reader can keep "drilling down" for more detail as needed.

Rational Rose, by Rational Software Corporation, is a visual software-modeling tool based on UML. "Rational Rose allows you to visualize, understand, and refine your requirements and architecture before committing them to code," the company claims on its Web site. In theory at least, this tool could be a boon to technical writers as well as to programmers. In fact, some companies have tried to use Rose as an API document-generating tool.

Whatever type of programming documentation you develop, you will be doing your programming readers a service if you model the documentation after the software itself.

However, judging by the personal experiences of some in the audience (at Andrea Ames' presentation during the March meeting), Rose offers more promise than fulfillment. The consensus, it seemed, was that by the time you overcome all the obstacles thrown in your path, you might as well have written your document from scratch.

Because the structure of your document is dictated by the structure of the software, you should "Let the programmers do the structural stuff, " Andrea says. Your job, then, is to do what you're best at: "Document the gotchas, write the how-to instructions and conceptual explanations, describe how classes work together and which classes can work with others."

In describing how she writes for programmers, Andrea describes hers as "a technology-versus-tools approach." Her strategy is that you, the writer, provide the tools, but not the technology. That is, your job is to contribute your writing, interviewing, and organization skills. It's also your job to obtain training about programming, but in the broadest possible manner. It is *not* your job to provide the subject details, for to do so would entail becoming a peer programmer. "Let your employer and the project provide the details. The details do not come from your learning," Andrea says.

References and Resources

"Get your facts first and then you can distort them as much as you wish."
— *Mark Twain (1835-1910)*

- Andrea Ames: andrea@verbal-imagery.com, (650) 365-7520
- Grady Booch: numerous titles, including *Object-Oriented Analysis and Design With Applications*; *Object Solutions: Managing the Object-Oriented Project*; *The Unified Modeling Language User Guide*; and *The Best of Booch: Designing Strategies for Object Technology*
- Martin Fowler with Kendall Scott: *UML Distilled*
- Ivar Jacobson: *Object-Oriented Software Engineering: A Use Case Driven Approach*
- Geoffrey James: *The Tao of Programming*
- Rational Software Corporation, Rational Rose (www.rational.com)
- James Rumbaugh: numerous titles, including *Object-Oriented Modeling and Design*; *The Unified Modeling Language Reference Manual*; and *The Unified Software Development Process*
- Sun Microsystems: www.sun.com; www.java.sun.com or www.javasoft.com (Java); www.java.sun.com/products/

Continues ☞

Writing for Programmers

Continued from page 8

javahelp/index.html (JavaHelp); www.java.sun.com/products/jdk/javadoc/index.html (JavaDoc)

- David A. Taylor: *Object Technology, A Manager's Guide* [Author's note: I found this book to be an excellent and succinct (200 pages) introduction into OOdom.]
- Writer's training: Santa Rosa JC, Sonoma State University (classroom); numerous "Teach Yourself" tutorials, including those for C++, Java, PERL, and VisualBasic (books); O'Reilly & Associates: www.oreilly.com; Smart Planet: www.smartplanet.com; WebMonkey, the Web Developers Resource: www.webmonkey.com (online)

Workshops in Technical Communication

at the School of Extended Education, Sonoma State University

"Introduction to Technical Writing"
Saturday, Sept. 30, 9am - 4pm
Instructor: Mark Weddle, Cisco Systems

"Researching Information Gathering and Analysis"
Saturday, Oct. 21, 9am - 1pm
Instructor: Eunice Malley, Next Level Communications

For more information, visit www.sonoma.edu/exed/Text/Fall/tc.html

Software Technical Writer

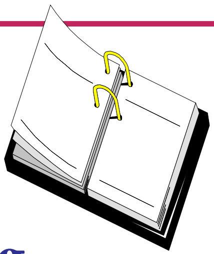
Next Level Communications, Rohnert Park

This position involves, but is not limited to, writing software installation and UNIX administration materials for our telecommunications application. Bring your interest in learning new technologies and in working hands-on with our equipment. We need someone to produce documentation that is accurate, grammatically correct, and adheres to our style guidelines.

This position requires the following.

- Three or more years of experience writing software documentation, targeted for various audiences
- B.A., B.S, or graduate degree in Technical Writing, Journalism, English, Computer Science, Engineering, or related area
- Experience proactively gathering technical information by interviewing developers and product managers, reading specifications; using the product; and attending product development meetings
- Ability to work on multiple projects simultaneously, and manage time and deliverables in a rapidly changing environment.
- FrameMaker experience
- Good knowledge of English grammar
- Technical knowledge of UNIX and Oracle is a plus.

Contact: Dena Sherick — dsherick@nlc.com



Coming Soon!

Aug. 27 to Sept. 1, 2000

Seybold Conference 2000 Moscone Center, San Francisco

Geared for web and print publishing professionals, Seybold features seminars on such topics as XML, PDF, and E-Books, and tutorials on Photoshop, Acrobat, XML, writing for the Web, and graphic design for the Web. The exposition (free admission) runs from 8/29 to 8/31 and is a golden opportunity to see the latest publishing tools and technology (and pick up some fun freebies too!). www.key3media.com/seyboldseminars/sf2000

October 19-21, 2000

STC Regions 7 and 8 Pan-Pacific Conference in Hawaii

For late-breaking information, see www.pan-pacific.org, and Jack Molisani's articles there.