THE DISSEMINATION OF TECHNICAL INFORMATION IN A SOFTWARE DEVELOPMENT ORGANIZATION

Timothy D. Gill

Member of Technical Staff

AT&T, UNIX® Computing Technology Laboratory

IEEE, ACM

ABSTRACT

The dissemination of technical information in a software development organization—project memos, software engineering process documents, local computing facility documentation—is often not organized and controlled adequately to meet the needs of staff. The inadequacies hamper project members in finding and using the information they need, lowering productivity and increasing costs. This paper describes common problems in this activity and presents a model of the development and use of documents in these organizations. Requirements for a simple system solving the problems are given, and LASO, a prototype based on the model and meeting most of the requirements, is described.

INTRODUCTION

The availability of appropriate technical information to the staff in a software development organization is important for productivity. Software development is a document-intensive, even document-driven, activity, and these documents undergo frequent revision. In addition, software development projects often depend on computing resources provided by internal organizations that must regularly make up-to-date technical information on the current computing environment available to their users. The ability to locate and access these documents and local sources of information, and to be confident that they are the relevant versions required, is a fundamental need.

Unfortunately, the development and dissemination of this technical information is often plagued with problems that prevent this need from being met and that contribute to lower productivity and higher overhead costs. The problems result partly from the overhead nature of such document development and the lack of adequate organization or control, and "getting the software out" usually takes a higher priority. Some are simply the by-product of the size and complexity of software development organizations: hundreds of people may work on numerous projects in one facility. The costs of this internal document development can be significant: "...for medium and large systems, paperwork is the second largest known cost element [after defect removal]... in the sum of all documents, over 200 pages per 1000 lines of code may be written..." (Ref. 1).

This paper presents a prototype system, LASO (for "Libraries for Accessing, Sharing, and Organizing" documents), that addresses these problems. The system was designed to meet an informal set of requirements developed from an analysis of the problems and from a model of the development and use of documents and technical information in a large R&D software organization. It was partially adapted from a previous system that provided a simple software project development environment for small projects (Ref. 2), and it builds on experience with similar problems gained in a variety of software development organizations in government, academia, and business.

Documents in this problem domain—those supporting the process of software development—fall at the low end of the length/printing-complexity spectrum for all printed material. Figure 1 presents an approximate representation of this spectrum; the rectangle at the left contains this domain.

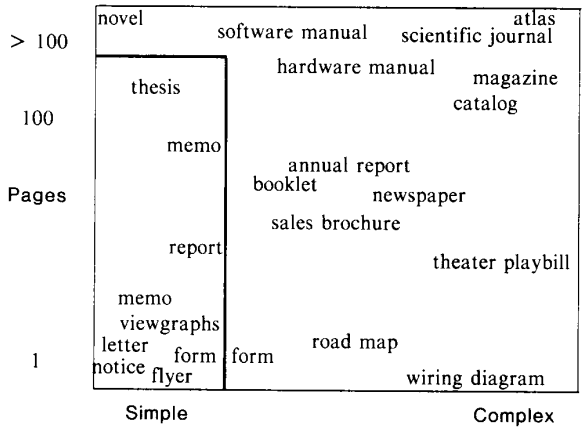

Figure 1. Range of Document Printing Complexity and Length

Typically, these documents are short (i.e., 5 to 50 or 100 pages, and even shorter for local computer facility information), require no complex printing or graphics, and are produced locally using available text processing software. In addition, they have neither the high quality requirements of the customer documentation for the software product itself nor the special requirements of online documentation systems (Ref. 3).

The LASO prototype is a simple system designed to support this low end of the document spectrum in software development organizations and to assist in solving the basic problems found there. It provides a modest but useful level of functionality that supports better document development and access and can be used for nearly all documents in the problem domain. Implemented in a UNIX® operating system environment (Ref. 4), it contains less than 2000 lines of UNIX system command language code; it uses the basic UNIX system utilities and a "batch" text processing system. The model and requirements on which LASO is designed, however, are not tied to any particular operating system; they provide a basis from which other solutions to the problems described might be built.

UNIX is a registered trademark of AT&T.

88CH2646-8/88/0000-0217 $1.00 Copyright 1988 IEEE

COMMON PROBLEMS

In developing a software system, specifying the functional requirements usually occurs first, in the traditional "waterfall model" (Ref. 5). The task of identifying and analyzing the problems is a preliminary part of that effort. In developing LASO, a detailed effort was made to identify the myriad problems that occur in the development and use of documents in software development organizations; each problem was to be specifically addressed by a system requirement. This section summarizes those problems.

As Fairley points out in Ref. 5, a certain set of software engineering process documents are considered the minimum necessary to support a software development project. The development, revision, and dissemination of this set of documents are subject to numerous problems. Similar problems occur as well for local computing facility documentation. The most common problems for both types of documents are in five areas:

- organization
- location
- reproduction and use
- timeliness
- version control

Problems summarized in this section do not always exist together, and some have been solved by one organization but not by another. However, anyone who has experienced any of the problems will recognize the frustration that each of them can cause.

The problems in organization all result from lack of an overall framework for such internal document development. For example, individuals create project documents but keep the source files in private, personal directories. The scope of a document is not clear: what organizational entity is it relevant to? You cannot find a summary document that describes the set of documents associated with a project or organization, or appropriate subsets of them. There is no uniform mechanism to identify documents or collections of documents: arbitrary filenames and directory trees of files are used but they can be confusing to persons other than the creator. Identification is especially problematic when information comes in so many forms—memos, handouts, text files, hand-written notes, n-th generation photocopies, and electronic mail or news announcements (which are often not captured in more permanent form).

Problems in finding documents are fundamentally caused by these organizational problems, but are aggravated by other factors. There is often no command available to locate a document through a simple search based on title, author, or specified keywords; if there is, it covers only a subset of existing documents. Often you cannot even determine if documents exist on a given topic. If one exists, you don't know where to look for it. Is it available online? If so, where? One underlying reason for problems in finding documents is that they are often distributed in printed form to a preset "Copy to" list known at the time of distribution, and yet additional persons need to obtain the document later.

Once you locate a document, you may encounter additional problems using it. If it doesn't exist online, you have to make your own photocopy, and you cannot search the text for keywords. If it does exist online, it is not in a form you can readily print: it's an unformatted text file that needs to be formatted "by hand" to print cleanly, or it's formatted in assorted inappropriate ways: no page numbers, wrong page length, wrong line length, wrong output device, etc. You want a typeset version but it exists only in a formatted ASCII form, or you want the source text (to reuse a paragraph) but cannot access it. If you are a new user or project member, you cannot easily get an initial (let alone complete!) set of documents relevant to your task or project.

If you manage to obtain a document you want in a printed or online form, you may find that you can't be sure it is the right one. The document lacks systematic identification, or the set it belongs to lacks a consistent appearance. You can't tell if it is up-to-date: the document may not even be dated, or if it is, it is dated with the date it was printed, not the date it was last revised; or it is dated just on the title page, but you only have tables from the appendix. No summary document exists to tell you what is the latest version of all documents. On a hectic project, you get two versions of the same document dated the same day on your desk and can't tell which is more recent.

Version control is often not done at all on documents, even if it is used for the software; this causes additional problems. You are asked to read multiple drafts of long documents but don't know what has changed, as the previous version is not available for comparison, or existing tools to show differences are not easy to use. You cannot obtain a previous version when you must, e.g., when the project decides to revert to an earlier design stage. If version control is used, the version number and revision date for a particular version are not printed on the document itself: typically, the version information printed on the document is not tied to the version control system at all but is left to the style of the author—so you get multiple versions marked "Issue 1" or "Draft 2," but these terms lack precise meaning.

A MODEL OF THE PROBLEM DOMAIN

In developing a software system an essential question is "What do we really want to build?" As Brooks reminds us (Ref. 6), establishing functional requirements is the hardest part of building software, and nothing cripples a system as much as getting the requirements wrong. A conceptual model is helpful in getting the requirements right. What are the objects in the problem domain? What are their attributes and relationships to one another? What are the fundamental characteristics of the problem domain? Can the model be relied on to answer a well-defined set of questions about the problem domain, to a reasonable tolerance?

Input for Constructing a Model

The LASO model of document development and use in a software development organization is simple, and is built from three sources: the objects in the problem domain, the fundamental characteristics of the problem domain, and common cultural experience.

Problem Domain Objects and Their Attributes

The objects and their relevant attributes identified for this problem domain are:

- individuals (name, organizational affiliation)
- organizational entities (hierarchical level, identifying number and name)

- projects (name, encompassing organizational entity)
- document collections (subject area, organizational scope, number of documents)
- documents (title, authors, current version, intended audience, file name/identification/ID)
- document versions (version identification, status, date, actual audience)
- logical document elements: sections, subsections, figures, tables, quotations, tables of contents, etc.
- text processing software (markup commands used)
- printers and display terminals (support for typeset printing or not)

Some of the identified attributes are not immediately clear. For example, a printed document always has an "ID" even if it is only a file name in an individual's collection of files; and each printed original is a "version" that has a status—that of "Draft" seems to be the most common—even if the printed version is made only to get a clean working copy. Once identified, however, these attributes play a part in obtaining a good solution.

Fundamental Characteristics of the Problem Domain

Some fundamental characteristics of the problem domain of document development and use in a software development organization are:

a. *varying stability of the information*: The lifetime of a single version of a document may vary considerably. There is a basic difference, however, between documents that are undergoing active development and those that are stable.

b. *varying user/author ratio*: The ratio of users of a document to the number of authors also varies. The basic difference is between a low user/author ratio (a small group of users, e.g., 10 or fewer) and a high user/author ratio (scores or hundreds of users).

c. *varying output formats*: Documents are printed with varying output formats: as memos, project documents, forms, reference guides, letters, meeting notes; with titles and authors; without titles and authors, etc. A basic difference exists between the markup commands (the embedded formatting commands used by batch text processing systems) commonly used by most documents for the body of text (for lists, displays, section headings, etc.) and the more varying markup commands used to control the overall document appearance and the look of the front matter (Ref. 7).

d. *role of hierarchies*: The intended audience of a document is related to the organizational hierarchies that exist, and the target audience of a document sometimes rises in the hierarchy over time. Document collections themselves can also be hierarchical.

e. *logical collections*: Documents fall into logical collections, by topic, project, audience, or purpose.

f. *basic actions*: Documents in active development are begun, written, read, revised, printed, and distributed. Stable documents are located, browsed, searched for keywords, reproduced, read, and used.

Some of these characteristics are related. For example, documents characterized by active development tend toward a low user/author ratio, while stable documents tend towards a high user/author ratio.

Borrowing from Common Cultural Experience

Common cultural experience provides a final source for constructing our model: book libraries. Everyone is familiar with libraries. The notions of different libraries, each with books and a catalog to help locate the books via an ID (e.g., a Library of Congress cataloging number), are readily borrowed for our model. The main differences in this problem domain are in number of elements and level of complexity: book libraries have far more of each.

The LASO Model

Some aspects of the LASO model of document development and use are shown in Figure 2.

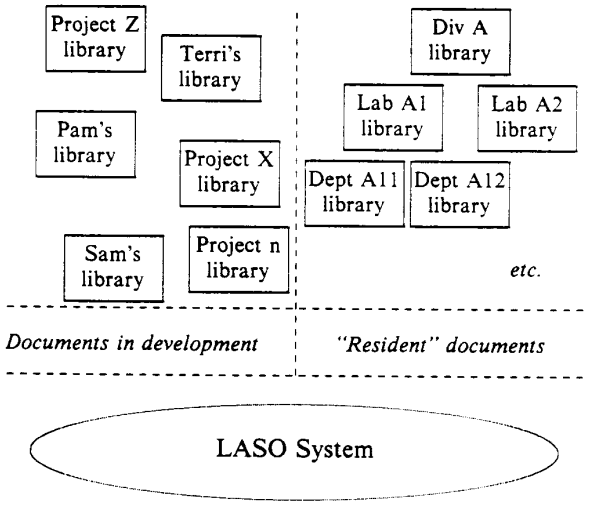

Figure 2. LASO Model of Document Development and Use

Although the model remains incomplete and needs refinement, its basic aspects are:

- Documents in a software development organization fall in two primary groups: project documents and general technical information documents.
  - ☐ One deliverable product of a software development project is the set of internal process documents: the project plan, functional requirements, architectural design, detailed design, test specs, etc. This set may be small (Ref. 2) or large (Ref. 8 describes a set of documents that define only the design of a software system). The more formal the development process used, the more critical this set of documents becomes and the more frequently it is used to assess the current status of the project.
  - ☐ General technical information documents include local computing facility information (e.g., using the printers, accessing the network) and process and procedural information for organizational entities (e.g., the code review process for a lab or the integration procedures for a department). Sometimes this information is in a form other than a document (electronic mail or news); however, it can be placed in a document, and the model assumes it is.

- Each individual, project, and organizational entity has a single library, with a unique *library ID*. Project libraries, departmental libraries, lab libraries, etc. may have an administrator, but personal libraries do not.
- Documents in a library have *document IDs* unique to that library, assigned by the administrator or owner of the library.
- Each library has a *catalog*, with a single entry containing descriptive information on each document (including at least the document ID and title).
- Libraries for organizational entities reflect the hierarchy of the organization: if the review procedures used by a department are adopted by the lab that encompasses that department, the relevant document migrates up to the lab library. A document belongs to the lowest reasonable library in the hierarchy of libraries that exist in the organization.
- All libraries support the active development of documents. In addition, a library can also function as a *resident library*, which means that it contains information of ongoing usefulness and its documents can be maintained in formatted as well as source text form. Resident libraries have documents with a high user/author ratio and moderate-to-high stability, and are usually associated with organizational entities such as departments. Libraries without the resident attribute contain only the source text form of documents and tend to have a low user/author ratio. Project documents are typically in a development-only library and local technical information documents are typically in a resident library.
- The documents in a library can be, if useful, logically grouped into *series*, a loosely related collection of documents covering a broad subject area, and *sets*, a closely related collection of documents covering a single topic. A set is part of a series; series may contain individual documents or sets of documents.
- Documents can be processed into different output formats without altering the source text, keeping the markup commands, even for front matter, unchanged.
- A single command can correspond to each basic action of the users in the problem domain.

SYSTEM REQUIREMENTS

Informal requirements were developed before building the LASO prototype. Individual requirements were specified to solve each of the known problems as well as to reflect the model just described. Examples of these requirements include support for multiple document libraries, immediate availability of documents for printing, keyword-searchable catalogs, printable catalogs, and mandatory version control.

Other requirements are worth mentioning specifically. The system must:

1. be able to handle all problem domain documents developed and used in the organization
2. allow, even encourage, very short documents (1–2 pages), which can provide easy dissemination of local technical information
3. allow documents to be available for browsing at a terminal as well as for printing
4. provide a mechanism to allow varied, customized, uniform, and unique output formats
5. integrate the version control mechanisms with the output formats used to print documents, so that the version number and date provided by the version control system are automatically printed on each page of a document
6. print the library ID and document ID on each page
7. force a new version to be created whenever a printed version of a document that is reserved for editing is requested by the person editing the document

This last requirement, not always popular, prevents two different printed versions from being identified as the same and was added when an earlier version failed to solve this problem without it.

THE LASO DOCUMENT LIBRARY SYSTEM

The LASO prototype is implemented as a set of UNIX system shell scripts. The UNIX system provides a fertile environment for this type of software because of its flexibility and rich set of tools. The two basic tools used by LASO are SCCS (the Source Code Control System) for version control and the Documenter's Workbench® (DWB) software for text processing; an earlier system on which LASO was based (Ref. 2) was implemented using RCS (the Revision Control System) and Scribe®.

The prototype was built not only to show the viability of the system and to test the validity of the LASO model, but to have a functioning system that could be used. The model needs to be refined and the prototype does not meet all the requirements, but both currently contribute to solving many of the problems previously described.

Features

The primary features of the current LASO prototype are:

- comprehensiveness (can be used for most documents)
- support for multiple libraries (personal, project, departmental, lab, division, etc.)
- support for resident libraries (formatted versions, both typeset and ASCII, of documents immediately available for display, searching, and printing)
- integration of SCCS version control with DWB processing of documents
- "add-on" implementation (LASO is an optional layer—it does not prevent users from using basic UNIX system commands on the library contents)
- administrator-only capabilities (e.g., maintaining a cataloging scheme) for non-personal libraries
- owner-controlled access to documents
- automatic "diffmarking" of two versions of a document, for display or printing

Design

The design goals of the prototype were:

Documenter's Workbench is a registered trademark of AT&T.
Scribe is a trademark of Unilogic Corporation.

- make it easy to organize, develop, find, use, and reuse documents at the low end of the length/printing-complexity spectrum (Fig. 1)
- meet 90% of local document development and access needs 90% of time
- make the system simple enough for a novice while still useful to experienced users
- keep source text files of documents fully processable on a stand-alone basis by the DWB macro package used, to support their portability
- minimize the setup and file overhead required to establish a library
- avoid UNIX system command line options
- allow commands to work on multiple documents
- use the power of UNIX system regular expressions and file permissions

Several aspects of the design are worth noting:

- An intermediate file of markup commands, called an *output format definition file* (described more fully below), is used between the DWB macro package and the source text file being formatted (Fig. 3). These files can be defined on system-wide, library, or individual document basis.
- A template or skeleton file is used in each library to begin new documents.
- Commands accept as arguments only document IDs and, where appropriate, string search patterns. Occasional prompts are used as necessary.

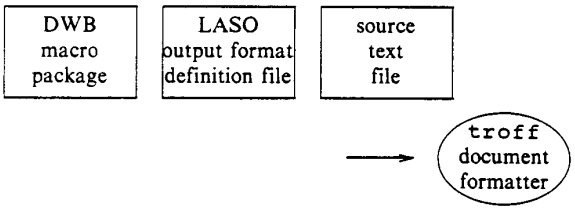

Figure 3. Use of LASO Output Format Definition Files

The output format definition file provides the mechanism by which the version control information of a version of a source file is incorporated into the formatted document. In addition, as its name suggests, it is the mechanism used to define different output formats without changing the document source text. The DWB text formatter, `troff`, uses macro packages as the means to define a set of markup commands; these macro packages can generally be characterized as "procedural markup" rather than "descriptive markup" (Ref. 9). The use of output format definition files, processed after the macro package is read, allows one to redefine selected macros (mostly those governing overall style and the appearance of front matter) as necessary to achieve the desired format; this design attempts to move the existing macros more in the direction of "descriptive markup."

LASO provides a set of output formats that may be used by any library. A library can have its own set of output formats, to provide for individual or project styles. In addition, an output format definition file named the same as a document ID allows unique formatting needs of that document to be defined outside the document source text itself.[1] And making the output format definition file empty causes the standard format provided by the DWB macro package to be used without modification.

The alternatives provided by these ways of defining output formats, and by the definition file itself, are an attempt to provide a flexible solution, in this batch text processing system, to the related design choice encountered in WYSIWYG text processors of whether document "style sheets" are stored with the documents they apply to or separately (Ref. 10). In addition, removing the more difficult part of macro coding to separate files that are common, rather than letting it be isolated in a single document, increases the reuse of this code.

Command Interface

The LASO commands fall in four groups: accessing resident versions of documents, version control, display and printing of documents under active development, and miscellaneous. The commands for accessing resident documents are:

Name	Use
`findDoc`	find document IDs and titles in library catalog, using keywords
`displayDoc`	display (typeset, if graphics terminal) documents on terminal
`printDoc`	print typeset documents on printer
`searchDoc`	search ASCII documents for strings

The commands for doing basic version control operations are:

Name	Use
`reserveS`	reserve source files for editing
`releaseS`	release reservation on source files
`depositS`	commit editing changes made to version control system
`inspectS`	obtain read-only copy of source files
`initializeS`	create new source files

The command names were adapted from the Gandalf Project at Carnegie-Mellon (Ref. 11). Note that a side benefit of the design of these commands is that they can be used as an interface for version control for files other than document source text (Ref. 2).

The commands for displaying and printing documents as they are developed are:

Name	Use
`displayS`	format (typeset) and display source text files on graphics terminal
`printS`	format (typeset) and print source text files on a printer

The other commands of LASO are used to administer a library or obtain information about it:

1. This paper itself was formatted for internal AT&T use using a standard AT&T memo format provided by LASO, but for the conference proceedings using a unique definition file associated with the document ID in my personal library.

Name	Use
`filesS`	list names of source files in library
`statusS`	list names of source files currently reserved from library
`historyS`	list version history of source files
`authorizeS`	change list of users authorized to edit source files
`installDoc`	format source text files into resident library output directories
`renumberDoc`	change the ID (filename) of documents
`deleteDoc`	delete documents from library

FUTURE EVOLUTION OF LASO

The current version of LASO is a prototype that is used by individuals, small projects, and a department, all of whom share the same mainframe. However, we work in a complex computing environment of mainframes, minicomputers, desktop minis, PCs, and technical workstations, connected in networks. An obvious direction for the system to evolve is to support the use of LASO document libraries in this networked environment. This would help realize the potential of the model to gain more control over and provide access to the hundreds of documents that are issued every month in our facility, duplicating the benefits that have already occurred on a smaller scale. One networked file sharing capability in use here is RFS, AT&T's UNIX System V Remote File Sharing facility, which allows files and sets of files on one machine to be made available for use by users on other machines in a transparent way; the user is not aware that the file is being accessed remotely. Given the LASO model of document libraries tied to organizational entities, I expect that adapting LASO for use in this networked environment will be a straightforward process of making departmental, lab, and division document libraries available as remote directories and ensuring appropriate administrative control.

A related direction for LASO evolution is a capability for doing a search on some set of document libraries instead of a single one at a time; this broad-based search will meet certain needs for searching that cannot be restricted to one library beforehand. Numerous other small enhancements are planned: tighter control over the "version status" of a document, to enforce project-specific values and clarify the jumble of terms now used ("draft," "preliminary," "review copy," "Issue *n*," "baselined," etc.); integration of landscape-mode tables into portrait-mode documents; a facility for inclusion of shared text modules themselves under version control; and additional LASO system output formats.

CONCLUSION

The existence of the problems discussed in this paper could be seen as a variation of the old line about the shoemaker's children going shoeless. Commercial document retrieval and full-text search systems exist but are often not applied to this small problem domain, for overhead, cost or other reasons. Hypertext systems (Refs. 12, 13) are beginning to be more widely explored and accepted, and will eventually be useful in this problem domain, but address the more complex issues of information use and not the basic problems described here. The problems are elementary and lend themselves to a simple solution that builds naturally on the elements of the problem domain. The LASO model and prototype constitute one possible solution requiring only a minor investment of resources, discipline, and administration.

ACKNOWLEDGMENTS

I would like to thank Jim Blinn, Michael Sabrio, Mark Hoffman, Helene Armitage, Audrey Kuna, and Sam Saal for their useful feedback on the prototype; Jim, Michael, Susan Pendleton, Lorraine Murray, and Lois Wolter for their detailed comments on this paper; and Ken Smith for permission to adapt his diagram.

REFERENCES

1. T. C. Jones, "Optimizing Software Productivity and Quality," Technology Transfer Institute, course notebook, Santa Monica, CA, 1988.
2. T. D. Gill, "A Simple Environment for Project Development under UNIX," Wang Institute of Graduate Studies Technical Report TR-85-22, Dec 1985.
3. J. H. Walker, "Issues and Strategies for Online Documentation," IEEE Transactions on Professional Communication, vol. PC 30, pp 235-248, Dec 1987.
4. UNIX® System V Release 3.1 Product Overview, select code 305-563, AT&T: Indianapolis, IN, 1987.
5. R.E. Fairley, Software Engineering Concepts, McGraw-Hill: New York, NY, 1985.
6. F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," IEEE Computer, vol. 20, pp 10-19, Apr 1987.
7. R. Furuta, J. Scofield, A. Shaw, "Document Formatting Systems: Survey, Concepts, Issues," ACM Computing Surveys, vol. 14, pp 417-472, Sep 1982.
8. S. D. Hester, D. L. Parnas, D. F. Utter, "Using Documentation as a Software Design Medium," Bell Labs Technical Journal, vol. 60, pp 1941-1977, Oct 1981.
9. J. H. Coombs, A. H. Renear, S. J. DeRose, "Markup Systems and the Future of Scholarly Text Processing," Communications of the ACM, vol. 30, pp 933-947, Nov 1987.
10. J. Johnson, R. J. Beach, "Styles in Document Editing Systems," IEEE Computer, vol. 21, pp 32-43, Jan 1988.
11. D. Notkin, "The Gandalf Project," Journal of Systems and Software, vol. 5, pp 91-105, May 1985.
12. J. Conklin, "Hypertext: An Introduction and Survey," IEEE Computer, vol.20, pp 17-40, Sep 1987.
13. J. H. Walker, "Supporting Document Development with Concordia," IEEE Computer, vol. 21, pp 48-59, Jan 1988.