ELECTRONIC PUBLISHING, VOL . 6(1), 23–34 (MARCH 1993)

Separation of concerns for indexing

DAVID ALEX LAMB AND MARGARET ANNE LAMB

Department of Computing and Information Science
Queen's University
Kingston, Ontario K7L 3N6, Canada

SUMMARY

Separation of concerns is a fundamental principle for managing conplex tasks. Previous tools for assisting in generating back-of-the-book indexes do not apply this principle as thoroughly as they might; in particular, most confuse two issues: recording where references occur in the main text, and deciding what terms should appear in the index. This paper describes a general facility for multi-level indexes that embodies this principle, usable in any document formatter that can produce a secondary output file recording page numbers where references occur. LATEX, `Scribe`, and `nroff`/`troff` fall in this category.

KEY WORDS Document preparation Indexes

1 INTRODUCTION

Developing large systems requires some principle for dividing complex tasks into simpler ones. One such principle is *separation of concerns*: one should avoid mingling separate concerns where possible. Software designers are accustomed to applying these principles at the functional component level; this paper argues that it is fruitful to separate concerns at the 'requirements' level, where one analyses what different clusters of user skills might apply to different aspects of the task at hand. We apply separation of concerns to the design of a tool for developing back-of-the-book indexes.

A large reference document such as a manual or a textbook needs an index to help readers quickly locate material on specific subjects. The indexing problem involves at least eight tasks, each requiring a different viewpoint or expertise.

1. Deciding what topics to index. This requires understanding the material and its intended audience; either a professional indexer or the author might do it.
2. Choosing the appearance of the index: fonts, point sizes, layout, and so on. It may involve classifying references, to separate primary discussions of topics from incidental mentions (boldfacing the former, for example). A book designer usually makes these decisions.
3. Deciding what terms to index. The person who does task 1 typically thinks of a primary term or phrase to index; task 3 involves imagining how typical readers will try to use the index to find material, thinking of synonymous terms, and inserting cross-references to other index entries. The expertise of the professional indexer is important here; authors may do well at task 1, but are often poor at task 3 [4] .

CCC 0894–3982/93/010023–12 *Received 24 June 1991*
Revised 1 March 1993

4. Deciding what regions in the text to index. Typically this requires understanding what the text is saying, and (depending on task 2) separating major discussions of a topic from incidental mentions of it. This task feeds back into task 1, since the indexer (or author) may discover additional topics while reading the text. It also requires deciding whether to reference a particular point in the text, a particular page, a range of pages, or an entire unit (such as a chapter or section).
5. Extracting a correspondence between textual regions and *references* that will appear in the index. Conventionally, such references are page number ranges; in advanced systems, they may be some representation of hypertext links. Depending on the decisions from task 2, each reference might require a classification as well. An indexer working from page proofs does this manually, but many document formatting systems can automate it.
6. Deciding how to sort the index. Indexers have sorting conventions such as treating ‘St’ as if spelled out ‘Saint’ , and ignoring certain words, such as prepositions. These rules are typically long-standing indexing conventions.
7. Routine clerical processing of the raw information from tasks 3, 4, and 5:
   a. sorting terms,
   b. merging references for the same entry,
   c. merging overlapping references (such as deleting a reference to page 3 when range 2–4 is already included).[1]

   These tasks are readily automated.
8. Typesetting the index. Today this is usually automated. However, someone (perhaps a programmer) must encode the results of task 2 for the text preparation tools.

As examples of these tasks, consider a piece of text that mentions Excalibur. In a collection of essays this may be a simple allusion, not worth indexing; in a treatise on famous names from literature, it would be an important indexing topic. This is a task 1 decision. Deciding to mark this particular piece of text is a task 4 decision. For task 3, the indexer might choose ‘Swords, Excalibur’ for people who wanted to look up famous swords but might not have thought of Excalibur, and ‘Arthur, see also Excalibur’ for people who thought of King Arthur first. For task 2, the book designer might choose to set the index in an eight-point font in two columns, to format entries of the form ‘Swords, Excalibur’ in two levels, and to use an indented rather than run-on style of subentry, leading to Figure 1.

```
Arthur                              ...
    ...                             Swords
    see also Excalibur                  Durendal        3, 27-31,
...                                         43-45
Excalibur            7, 65              Excalibur       7, 65
```

Figure 1. Sample multi-level index

[1] As discussed later, merging successive references into intervals (such as changing 1,2,3 to 1–3), requires a human decision.

Typical document formatters provide little support for generating such indexes, leading many people to design auxiliary programs for this purpose. In the early 1980s when we began to look at indexing, available indexing tools had several limitations, many of which could have been foreseen and avoided by applying user-centred separation of concerns. Most tools provide an operation that takes a phrase to appear in the index. The tools supply a page number, sort all the phrases, eliminate duplicates, combine page numbers for the same term, and format the term. A typical approach might require

```
.iX Excalibur
.iX Swords, Excalibur
```

at each reference to Excalibur.[2] The most important problem with this approach is a failure to separate task 4 from task 3: all the index entries must appear at each place that references them, which means that many changes require finding and modifying all previous occurrences of a term. For example:

- Adding index entries for synonyms or alternative search phrases, such as adding ‘Swords, Excalibur’ having first indexed only ‘Excalibur’
- Changing the appearance of index entries, such as italicizing the name *Excalibur*, or capitalizing the first words of index entries
- Dropping a particular term to form an abbreviated index.

Older tools also sometimes failed to separate task 6 from task 7.a, requiring that index terms sort in simple lexicographic order. In addition, many tools have several other incidental limitations:

- No provision for pageless entries, such as ‘see also’
- No provision for multi-level indexes, such as the Swords example

Section 3 surveys the indexing literature from the perspective of separation of concerns. For comparison purposes we first describe our own approach to indexing (Section 2), which we embodied in a prototype tool collection.

2 OVERVIEW OF `index`

This section describes our indexing program, `index`, and some companion tools (`preindex` and `dsplit`) that embody separation of concerns. They act as pre- and postprocessors to the document formatter; this paper describes how they interact with `nroff` and `troff`, but they can work with any document formatter that can produce an auxiliary file containing information about what regions of the output file reference indexed material.

Figure 2 outlines their operation. The indexer places marks, such as

```
.IX excal
```

in the `troff` source for the document. Separately, in the .idb database, she places directives that can be interpreted as saying “for all page references to the keyword ‘excal’,

[2] This happens to be the syntax Gehani [11] describes for UNIX `nroff`/`troff`-based formatting tools. Bentley and Kernighan [5] describe a more flexible collection of UNIX tools that could be adapted to handle this issue.

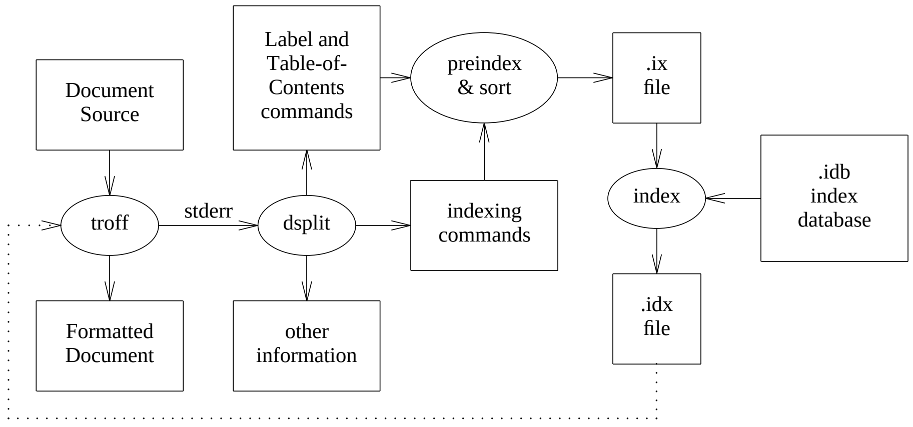

Figure 2. Operation of indexing tools

index the terms ‘Excalibur’ and ‘Swords, Excalibur’ ”. The tools produce a `troff` source file (the .idx file) for the index; it might be formatted separately, or fed back into the next formatting of the main document (dotted line).

There are four kinds of indexing mark:

- .IX is the conventional type of mark, taking a key as a parameter and denoting ‘index this key as occurring on the output page corresponding to this point in the text.’

The remaining marks potentially denote ranges of pages, addressing the separation of concerns 4 and 5:

- .SX takes a key as a parameter, and generates an entry for the range of pages corresponding to the entire current section. This requires extracting information from the table of contents about where sections start and stop, which in turn requires distinguishing whether section K+1 starts at the top of page P, to tell whether section K stops on page P or P−1. Customized section-heading macros record this information.
- .HX takes a key and a label as a parameter, and generates an entry for the range of pages from the label to the place where the .HX occurs. This requires cooperation from a separate cross-reference tool that manages labels and forward references; it is simpler in `Scribe` and LaTeX, which have built-in cross-referencing facilities.
- .LX takes a keyword and two labels as parameters, and references all the pages between the labels. This is rarely necessary (.HX being more common).

The `preindex` program turns the data these marks generate into entries of the form

key page$_1$ page$_2$ tag

which was designed to capture the essential information for task 5: *keyword* is the first argument to the indexing marks, the pages denote the region of the formatted document, and *tag* classifies the reference.

The .idb file to a first approximation is a simple map from keys to sets of index terms.

An index term is a sequence of phrases, each of which corresponds to a level in a multi-level index. Thus our previous Excalibur example would involve:[3]

```
excal => "Excalibur" ;
         "Swords"  "Excalibur"
```

to describe the two index terms for key 'excal'. The file format allows for some forms of cross-reference between terms, and for recording how to sort each term when necessary:

```
excal => exname: "\fIExcalibur\fP" sorted "Excalibur" ;
       "Swords" exname^
```

The `sorted` annotation specifies a sort key for the preceding term, which contains embedded *troff* formatting directives. The cross-reference definition (`exname:`) and its use (`exname^`) permit a single point of definition for how to format and sort the name 'Excalibur'.

The .idx output file is a series of `troff` commands for producing the index. For flexibility, the commands are macros that users can redefine. There are two commands:

.IE K "phrase" "pages"

means 'format a level K index entry from the given phrase and page range.' The default version of .IE uses the number K to set up appropriate indenting for the entry. .I0 takes no arguments; it marks a change in the first letter of a top-level index entry. The default version inserts an extra blank line as a separator.

Index terms in the .idb file are tagged to simplify certain automatable transformations. The only currently implemented tag, `strip`, specifies that the sort key is obtained by stripping out formatting directives. This mechanism allows for future expansion:

- Lynch and Petrie [18] describe six rules for generating articulated subject indexes; processing a `permute` tag could incorporate their rules.
- Jones [15] points out that, in addition to ignoring formatting codes and case distinctions, and treating abbreviations as though spelled out, computerized sorting needs to ignore punctuation marks, and in some fields, ignore entire prefixes, such as *para* and *meta* in chemistry.

It is unclear how often such transformations occur in current practice; the availability of tools to make them easy would probably increase their frequency.

For indexers working from page proofs, a format such as

page$_1$
  keyword$_1$
  keyword$_2$
  keyword$_3$ page$_2$
page$_3$
  keyword$_4$

requires less typing than .ix files. Here one gives a page number, then lists all those references that fall, or begin, on that page. The occurrence of page$_2$ means that

[3] Our prototype uses an idiosyncratic format (dictated by the tools we used to build it) for this information. In this paper we use one that easier to understand, and that we plan to use in a revised tool.

keyword$_3$ occupies page range page$_1$–page$_2$. Several systems aimed at professional indexers use such a format [10, 13, 14] (but with full index entries instead of keywords). We have an *awk* script to translate this format into an .ix file. Producing an index then requires running only a sort, the `index` program, and the text formatter.

3 RELATED WORK

The literature divides primarily into research systems and commercial products. The dividing line is hard to draw; for this paper, the distinction is in whether the reference focuses primarily on the lessons to be learned from designing the systems (research), or focuses on available features (commercial).

Jones [15] summarizes the state of computerized indexing in 1986. He surveys work on automated methods for extracting words from texts, but says they are still inadequate for indexes. Raper [20] says that with computer tools for clerical tasks, the unaided tasks of reading text and creating and checking index entries now takes 81% of his time instead of 60%; presuming these tasks still take the same amount of time as before, computer tools eliminated about 25% of the time to produce an index.

3.1 Research efforts

Gardner and Gardner [10] give their own separation of concerns for indexing:

- Choosing entries for the index. They recognize that the phrases in the index may need to differ from those that appear in the main text, and this requires human judgement.
- Sorting the index. They say there are no generally accepted standards, even whether to sort word-by-word or letter-by-letter (Mulvaney [19] cites the *Chicago Manual of Style*, which prefers letter by letter. She points out that many computer-based systems sort in ASCII collating sequence, which is unacceptable to professional indexers). Roman numerals must sort as the numbers they represent.
- Typesetting and printing the index.

Their tools require a hand-generated file; each section corresponds to a particular page, and lists all the phrases to index on that page. They allow up to four levels of entry.

Bentley and Kernighan [5] describe a collection of UNIX-based tools for indexing. In accordance with the UNIX philosophy, they provide a collection of separate filters for permuting terms, compressing runs of references, and sorting; indexers can combine these programs via pipes and shell scripts. They extract information from `troff` or `TEX` as we do, but give the text to be indexed as the argument, instead of using a tag. To control sorting, they allow

.ix *index term* %key *sort key*

As with any scheme that directly embeds index terms in the document, this requires a document-wide substitution if one changes either the original term or the sort key; however, it would be straightforward to add a step to their pipeline that translates keys into sets of index terms. For page range references they provide

```
.ix %begin index term
...
.ix %end index term
```

As we do, they banish details of index format to document formatter macros. They sketch methods for extending their tools to handle cross-references and one level of subentry. They claim to handle about 90% of what one wants with about 200 lines of `awk` code, and suggest that conventional programming techniques gain another 9% at a cost of at least a factor of 10 in program size. Our programs consist of 4092 lines of C code, excluding library modules, which tends to support their claim.

Chen and Harrison [7] describe a formatter-independent approach to indexing, and their `MakeIndex` tool. They discuss both source language approaches (such as `Scribe`, LaTeX, and `troff`) and direct-manipulation systems (such as `FrameMaker`, `Ventura Publisher`, and `Microsoft Word`). They discuss subindexing, sorting by keys distinct from the actual fields, cross-referencing, and merging. They parameterize the index tool's input and output for different formatters; we require a single simple input and output format, and banish adapting for different formatters to separate preprocessors and postprocessors (we use `nroff` to change calls to the formatting macros into appropriate LaTeX macro calls, for example). They provide control over whether to merge references, pointing out that merging is incorrect if each page discusses the subject only intermittently; `preindex` could (but does not yet) distinguish single-point references (.IX) from page range references (.SX, .HX, and .LX).

Abe and Berry [1] address the problem of having to 'flood' the document to be indexed with indexing-specific commands, which can make the original source hard to read. They rely on scanning for phrases that occur in the document, but have directives (actually, several separate files, each for one type of directive) for indexing a phrase from the document as a distinct phrase, and grouping subentries under a main entry. They prefer to use the `findphrases` program, which searches for repeated phrases, to form the initial list of index terms, but an indexer can edit this list. Our approach separates the index terms from the document, but avoids any direct tie between phrases in the text and phrases in the index; we do require a 'flood' of tags in the original (though less of one than systems that require tagging with full index entries, and that lack methods for indexing ranges of text). To find page information, they cleverly scan the output files from `ditroff`, which contain end-of-word, end-of-line, and end-of-page markers; though we have tailored our approach to `troff`, we have also easily adapted it to LaTeX, and could do so for other formatters. Their current implementation supports only word-by-word sorting, but it should be straightforward to extend (to, for example, ignore prepositions, or sort 'St' as 'Saint'). Indexing several distinct phrases wherever one of them occurs requires grouping or cross-references; thus they can achieve

```
Excalibur  6-10      and    Slashing see sword-swinging    but not    Slashing 6-10
Swords                      Sword-swinging 6-10                       Sword-swinging 6-10
  Excalibur 6-10
```

To be fair, their approach conforms to standard manual indexing practices, and it seems feasible to extend their program with another type of directive to allow this. Their scheme suffers from several problems that seem inherent in any scheme that automatically scans the document for phrases:

- Manual intervention to delete unwanted references (where an index term occurs in the document, but the indexer does not wish to include it in the index).
- Manual intervention to merge consecutive references (such as turning 1,2,3 into 1–3).[4]
- Altering the phrase search file to trick the phrase finder into doing the right thing. For example, if an index phrase spreads across two pages, they must search for two other phrases to force both page references to occur; we would use a page range reference (.HX or .LX) instead.

Any change to the text requires redoing these manual steps.

In a 1985 USENET news article (group net.sources), Lee Moore described how he produced a multi-level index for the University of Rochester Computer Science Department graduate handbook. He used the same technique we do for extracting page information from `troff`. He did not separate the marking of index points from the text of what goes into the index; typical index requests in his scheme look like

```
.IX "major heading"
.IX "topic with subtopics" "subtopic"
```

3.2 Commercial products

Descriptions of commercial products tend to concentrate on features; because of competitive pressure, they are likely to evolve rapidly. Thus the descriptions of the following products are reasonably accurate as of the time of the writing of the references, but may have become obsolete by now. The names of most of these products are trademarks of the companies that market them.

Fetters [9] evaluates nine commercial indexing products for the IBM PC: Cindex, IndexAid2, Indexer's Assistant, Indexit, INDEXX, In>Sort/DOS, Macrex, NLCindex, and wINDEX. For researchers, the survey is especially interesting for the cultural separation it reveals between the research community and the professional indexer. Her comparison chart lists 33 features important to professionals, grouped into the major categories of 'editing and displaying', 'formatting and printing', and 'sorting'. Most researchers (we believe) would consider the first two categories to be primarily the responsibility of separate editing and text formatting programs, respectively. For the professional, all the features must be well integrated, and so batch-style processing and separation of facilities into different tools, typical of research systems, is unacceptable.

Mulvaney [19] describes deficiencies of five existing commercial systems. Of particular interest is her mention of the need to 'flip' sub-entries with main entries, and to make global search and replace commands on the index text separately from the main text. Both are supported by our separation of tags from index text, and by our allowing several distinct index entries from one tag. She also discusses the need to separate the sort key from the text appearing in the index: ignoring articles and prepositions in subentries; omitting certain words (such as alpha- and beta- prefixes in chemical terms); and specifying alternative sort keys for some words (such as sorting '90' as if spelled out as 'ninety'.

[4] As they point out, manual intervention is clearly required at some level, to distinguish intermittent references from continued discussion. The issue is whether postprocessing the output from the scanner is the right place to do this.

Coverage of these issues by the commercial systems Fetters surveys is spotty; of the research systems, ours supports it (via optional `sorted` attributes for index entries), as do those of Bentley and Kernighan [5] and Chen and Harrison [7]. Mulvaney also mentions the need to support both indented and run-in styles of entry, and user-definable indentation; the batch-oriented research systems (including ours) can accommodate this by changing text formatter commands or macros.

3.3 Other work

There is a vast literature on automated indexing for information retrieval systems, much too large to survey in this paper. Most writers on the subject of back-of-the-book indexes (some of whom are professional indexers) assert that fully automatic methods are not yet good enough, and that human judgement is still necessary. However, Salton [21] reports that automated methods, based on natural language parsing, are improving; he gives an example where 83% of the generated phrases are appropriate.

The *amanuensis approach* is one way to combine computing power with human judgements: the computer amanuensis (a Greek word for a slave-scribe) automates algorithmic clerical tasks (such as searching a thesaurus), and makes suggestions for the human to accept or reject for heuristic tasks (such as suggesting phrases to include in the index). Kay [16] outlines this approach for the somewhat related (but harder) problem of machine-aided translation of natural language.

Separating the database of index terms simplifies building tools to help manage the terms themselves. One could build the database using a thesaurus of related terms. Dillon [8] describes an experiment in automatic book indexing with a manually created thesaurus. Thesaurus construction was about as difficult as indexing the book by hand, but the result may have been reusable for other books on similar topics. The resulting index performed well by the usual information retrieval criteria of precision and recall.

Wall [24] describes useful relationships in a thesaurus manager, such as ‘use/use for’ (substitute one phrase for another), ‘narrower term/broader term’ (which could guide grouping related terms), and ‘related term’ (which could guide creation of *see* and *see also* entries). A thesaurus would be especially useful to a group attempting to index a collection of related documents, to maintain consistency. Connecting the thesaurus with an indexing program could yield statistics on frequency of use of index terms. Abe and Berry’s [1] *phrase* file is a form of thesaurus; their *combine-phrase* file records ‘use’ relations; their *see*, *see under*, and *see also* are more precise versions of ‘related term’ and ‘broader term .

Barnes *et al*. [3] describes an experiment using the `SLC-II` system to automate indexing of a collection of scientific abstracts on isotope separation. Their system has several facilities that would be useful in an amanuensis, such as

- Morphological analysis of words to isolate affixes (suffixes and prefixes), reducing the number of distinct words the indexer must consider.
- Contextual information, such as ‘scope notes’, which are usually natural language explanations of terms. `SLC-II` formalizes some scope notes; one can record information such as “if you see the term ‘reactor wastes’ in a sentence with ‘extract’, ‘process’, ‘reduce’, or ‘treat’, index it as ‘radioactive waste processing’ ”.

4 CONCLUSION

This work has contributed the separation of concerns with which the paper began, particularly the separation of indexing points in the document from the database of terms to appear in the index. Bell and Suggate [4] emphasize the constant need to cross-check the growing index with the text to produce a high-quality index, and to ensure consistency between different ways of indexing a concept (such as ensuring that 'Excalibur' and 'Swords, Excalibur' get the same set of page references); the separation of concerns, and grouping several index entries under the same tag, address this issue.

Secondary contributions include

- Support for multi-document indexes (by tagging references in the separate .ix files, then merging into a single file).
- Support for abstractions of index entries, such as allowing several entries under the same key, and the 'strip' tag.
- Support for separating the sort key from the text that appears in the index; Bentley and Kernighan [5], Chen and Harrison [7], and several commercial systems [9] also support this.
- Support for arbitrary levels of subentry.
- Support for higher-level references, such as those to a section (.SX) or arbitrary range (.HX, .LX); Bentley and Kernighan [5]'s .ix %begin/.ix %end also supports this.

The prototype is adequate but not particularly interesting in itself, aside from its embodying these ideas. We built it to index a software engineering textbook [17], and have since used it for other documents. Executable binaries are available for SUN3 systems; we have not yet packaged the source code for export, primarily because we built it using tools that are not widely available.

Another conclusion from this work is the need for document formatting systems to provide well-designed mechanisms for extracting information about the formatted documents. `Scribe` and LaTeX provide direct mechanisms for getting auxiliary information about the document; `nroff` provides a clumsy mechanism. None of the three gave an easy way to determine if a section began at the top of a page; all required rewriting the section-starting macros. Other document formatters provide no way to get such information, and so eliminate the possibility of writing auxiliary programs to produce indexes. Since no document formatter can meet all the needs of its users, designers of such programs should consider incorporating some facility for generating auxiliary files.

4.1 Future work

The current plans for `index` are

- Package the tools for porting to non-SUN3 systems, which must await changes in the tools we used to build them.
- Build a set of index editing facilities as macros within the GNU Emacs editor [23]. It has the multi-window facilities Anderson [2] considered important, and facilities for expanding abbreviations as recommended by Hines and Winkel [14].

- Incorporate several indexing conventions and cross-checks that Borko and Bernier [6] describe:
  — If there is a *see* entry

  Slashing *see* sword-swinging

  then the 'sword-swinging' entry needs a parenthetical gloss:

  Sword-swinging (*slashing;* ...) 6–10

  — A broad page range reference (such as 11–35), or a long list of references, may signal index entries that are too general; the reader must flip through many pages to determine whether the references are relevant. An index program might optionally warn of such potential problems.

The approach is applicable to modern markup languages. In SGML, for example, it seems best to mark both the beginning and end of a section of text associated with a particular keyword. This eliminates the distinction among .IX, .SX, .LX, and .HX entries. If an SGML formatter can produce information in the format of .ix files, most of our tools can be reused.

For a what-you-see-is-what-you-get (WYSIWYG) formatter, it seems desirable to let the indexer select an arbitrary range of text, and associate that range with a particular keyword. Having information about the beginning and end of each reference, such a tool could automatically adjust index page ranges after most editing operations, including those that split a former single-page reference across two (or more) pages. Editing that eliminated one end of the marked region would require dialogue to adjust the range of marked text. It would likely still be desirable to regenerate the index on request, rather than automatically.

ACKNOWLEDGEMENTS

We first heard of the idea of separating the marking of index points from the text of what goes into an index from Bill Wulf, reporting what Paul Hilfinger had done to produce the index for an undergraduate textbook [25]. Our first experience with the technique of postprocessing auxiliary files from `Scribe` occurred when David Lamb produced the multiple indexes of the first DIANA reference manual [12]; it seems likely that someone else must have used this idea before 1981. He also wrote an earlier version of the facility described in this paper while working for Tartan Laboratories, Pittsburgh, Pennsylvania during 1983–4; Joseph M. Newcomer gave much useful advice on its design. In a private communication (June 1985), Art Evans reported that Tartan has since used it to generate indexes for Tartan internal manuals; the indexes were so large that the version of `Scribe` he had could not handle them. We wrote `index` and `preindex` during 1986 and 1987, using an early version of the IDL Toolkit [22] to manage the index database.

Mary-Ellen Maybee did much of the library research for this paper. Nancy Mulvaney raised our level of awareness of the needs of professional indexers, and pointed us to several of the references on indexing software. Robert Crawford, Richard Furuta, Ian Macleod, Gary Marchionini, and the anonymous referees gave helpful feedback on earlier drafts. The preparation of this report was sponsored in part by the Natural Sciences and Engineering Research Council of Canada under grant OPG000908, and in part by the Institute for Advanced Computer Studies at the University of Maryland.

REFERENCES

1. Kris K. Abe and Daniel M. Berry, ‘`indx` and `findphrases`: A system for generating indexes for ditroff documents’, *Software: Practice and Experience*, **19** (1), 1–34 (1989).
2. Charles Anderson, ‘<<ANSWER>>: an ‘off-the-shelf’ program for computer-aided indexing’, *The Indexer*, **13** (4), 236–238 (1983).
3. C. I. Barnes, L. Costantini, and S. Perschke, ‘Automatic indexing using the SLC-II system’, *Information Processing & Management*, **14** (2), 107–119 (1978).
4. Hazel Bell and Kate Suggate, ‘Computer-assisted indexes: two results assessed’, *The Indexer*, **14** (2), 95–98 (1984).
5. Jon L. Bentley and Brian W. Kernighan, ‘Tools for printing indexes’, *Electronic Publishing*, **1** (1), 3–17 (1988).
6. Harold Borko and Charles L. Bernier, *Indexing Concepts and Methods*, Academic Press, New York, 1978.
7. Pehong Chen and Michael A. Harrison, ‘Index preparation and processing’, *Software: Practice and Experience*, **18** (9), 897–915 (1988).
8. Martin Dillon, ‘Thesaurus-based automatic book indexing’, *Information Processing & Management*, **18** (4), 167–178 (1982).
9. Linda K. Fetters, *A Guide to Indexing Software*, fourth edition, American Society of Indexers, Inc., 1992. ISBN 0-936547-15-4
10. Ron Gardner and Eve Gardner, ‘Computer-aided indexing with SPITBOL and TEXTFORM®’, *The Indexer*, **13** (2), 115–119 (1982).
11. N. Gehani, *Document Formatting and Typesetting on the UNIX System*, Silicon Press, Summit, NJ, 1986.
12. G. Goos and W. A. Wulf, ‘Diana reference manual’, CMU-CS-81-101, Computer Science Department, Carnegie-Mellon University (March 1981).
13. Theodore C. Hines and Jessica L. Harris, ‘Computer-aided production of book indexes’, *The Indexer*, **7** (2), 49–54 (1970).
14. Theodore C. Hines and Lois Winkel, ‘Microcomputer-aided production of indices’, *The Indexer*, **11** (4), 198–201 (1979).
15. Kevin P. Jones, ‘Getting started in computerized indexing’, *The Indexer*, **15** (1), 9–13 (1986).
16. Martin Kay, ‘The proper place of man and machines in language translation’, CSL-80-11, Xerox Palo Alto Research Center (October 1980).
17. David Alex Lamb, *Software Engineering: Planning for Change*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
18. M. F. Lynch and J. H. Petrie, ‘A program suite for the production of articulated subject indexes’, *Computer Journal*, **16** (1), 46–51 (1973).
19. Nancy C. Mulvaney, ‘Software tools for indexing: what we need’, *The Indexer*, **17** (2), 108–113 (1990).
20. Richard Raper, ‘The business of computer-aided indexing’, *The Indexer*, **14** (2), 118–119 (1984).
21. Gerald Salton, ‘Automatic text indexing using complex identifiers’, in *ACM Conference on Document Processing Systems*, pp. 135–144, December 5–9, 1988.
22. Richard Snodgrass, *The Interface Description Language: Definition and Use*, Computer Science Press, Rockville, MD, 1989.
23. Richard M. Stallman, *GNU Emacs Manual*, Free Software Foundation, Cambridge, MA, October 1986. 675 Mass Ave., 02139. Fifth edition. GNU Emacs Version 18.
24. Eugene Wall, ‘Symbiotic development of thesauri and information systems: A case study’, *Journal of the American Society for Information Science*, **32** (1), 71–79 (1981).
25. W. A. Wulf, M. Shaw, P. N. Hilfinger, and L. Flon, *Fundamental Structures of Computer Science*, Addison-Wesley, 1981.