Document Concordance Generator

Design Specification

Eric Brickner

Chris Blanchard

20-260-495
Software Engineering Lab
Professor Carter
Winter, 1992

1. Scope

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm that is used. The design step produces a data design, an architectural design and a procedural design. The *data design* transforms the information domain model created during analysis into the data structures that will be required to implement the software. The *architectural design* defines the relationship among major structural components of the program . The *procedural design* transforms structural components into a procedural description of the software.

- R.Pressman

1.1 System Objectives

The overall objective of the Document Concordance Generator is to provide the user with a fast, easy-to-use tool to aid in the study of massive works of literature.

1.2 Hardware, Software and Human Interface

The DCG operates in a command line environment. Therefore, the primary interface of this project is between the software and the display. The display is used extensively in error messaging.

1.3 Major Software Function

The major function of the Document Concordance Generator is to generate a concordance of a specified input text file. The DCG has been decomposed into five functional modules, whose component functions are outlined below.

1. User Command Parser:
   - Gets Command Line
   - Checks if Parameters Valid - Counts Parameters
   - Opens Input File - Checks if able to open file
   - Opens Output File - Checks if exists, Creates Output File
   - Create Skipword List - Read in Skip Words from file, Add Nodes to List
   - Handles Errors - Display Error Message

2. Line/Word Parser:
   - Parses Lines - Gets Lines, Counts Line Numbers, Counts Page Numbers
   - Parses Words - Gets Words, Checks for End-Of-Line
   - Checks if Word Valid - Calls SkipWord

3. Skip Word Module:
   - Checks if Word is in List
   - Returns Status

4. Structure Builder
   - Adds Word Nodes
   - Adds Appearance Nodes
   - Searches Concordance Structure
5. Output Formatter
   - Creates Title Page
   - Creates Concordance Pages

1.4 Externally Defined Database

Input Text Document:

The Input Text Document is the input file to the DCG. This is an ascii text file whose lines are delimited by carriage returns and whose words are delimited by blank space and punctuation marks including commas, colons, quotation marks, and semicolons. or simplicity, words that are continued between lines shall be treated as single words, with the hyphen appearing in the final concordance.

Output Text Document:

The Output Text Document is the output file of the DCG. This is an ascii text file which contains a concordance of the Input Text Document. This file shall contain a header at the top of each page, a footer at the bottom of each page with page numbers, and a title page showing the name of the input file. For an example of the title page and a body page of the Output Text Document, please refer to Appendix A - Sample Output Text Document.

1.5 Major Design Constraints and Limitations

There are no constraints within the DCG that dictate the maximum size of either the internal data structures or of the input and output files. The size limitations on these items is imposed by the hardware environment in which the DCG operates.

2. Reference Documents

2.1 Existing Software Documentation

Blanchard, C. and Brickner, E. Document Concordance Generator: Software Project Plan. Unpublished. 1992.

Blanchard, C. and Brickner, E. Document Concordance Generator: Software Requirements Specification. Unpublished. 1992.

2.2 System Documentation

Heslop, B. and Angell, D. Mastering SunOS. Sybex, San Francisco. 1990.

2.3 Vendor Documents

Not applicable.

2.4 Technical Reference

Conn, Richard. CS Parts, Version 2. Unpublished. 1991.

Naiditch, D. Rendezvous with Ada: a programmer's introduction. John Wiley and Sons, New York. 1989.

Press, W. H. Numerical Recipes in C. Cambridge University Press, New York. 1990.

Pressman, R. Software Engineering: a practitioner's approach. McGraw-Hill, New York. 1992.

Sedgewick, R. Algorithms in C. Addison-Wesley Publishing Co., New York. 1990.

3. Design Description

3.1 Data Description

3.1.1 Review of Data Flow

Data Flow Diagram - Level 0

In the context diagram (figure 3.1), the DCG may be seen to receive input in the form of Input Text from the Input Text Document. The DCG also receives input from the Console. Output reaches the user in two ways: first, by means of the display, and second, by means of the Output Text Document.

Data Flow Diagram - Level 1

The level 1 DFD (figure 3.2) for the DCG may best be explained in terms of its components.

1.) The User Command Line flows from the Console to the User Command Parser, which then sends an Input Filename to the Line/Word Parser. Error Messages may also be sent to the Display. This module also sends the Output Filename to the Output Formatter.

2.) The Line/Word Parser accepts Input Text from the Input Text Document, and send Words to the Skip-Word Module and the Structure Builder, as well as Error Messages to the Display.

3.) The Skip-Word Module receives Words from the Line/Word Parser Module, and returns Status to that module.

4.) The Structure Builder sends Error Messages to the Display, and sends a Concordance Structure to the Output Formatter Module.

5.) The Output Formatter Module sends Error Messages to the Display, and Output Text to the Output Text Document.

Data Flow Diagram - Level 0

µ §

Figure 3.1

Data Flow Diagram - Level 1

µ §

Figure 3.2

3.1.2 Review of Data Structure

name: **Concordance Structure**
alias: CS

where used?
In two places, the Structure Builder places Words and their Coordinates of Appearance into the CS, and the Output Formatter retrieves Words and their Coordinates of Appearance from the CS.

how used?
Used to hold all Words and their Coordinates of Appearance.

content description?

A list of all non-skip Words within the Input Text Document and their associated Coordinates of Appearance

name: **Coordinates of Appearance**
alias: none

where used?
1.) Passed from the Line/Word Parser Module to the Structure Builder.
2.) Passed from the Structure Builder to the Concordance Structure.
3.) Retrieved from the Concordance Structure by the Output Formatter.

how used?
Used to specify the location (Page,Line) of a Word within the Text Input Document..

content description?
1.) Page Number
2.) Line Number

name: **Error Message**
alias: Error Msg

where used?
1.) Passed from the User Command Parser to the Display.
2.) Passed from the Line/Word Parser to the Display.
3.) Passed from the Structure Builder to the Display.
4.) Passed from the Output Formatter to the Display.

how used?
Used to communicate error conditions to the user.

content description?
A string of characters.

name: **Input Filename**
alias: none

where used?
Passed from the User Command Parser to the Line/Word Parser.

how used?
Used to specify which file to open for Input Text.

content description?
A string of characters.

name: **Input Text**
alias: none

where used?
Passed from the Input Text Document to the Line/Word Parser.

how used?
Used as input to the Line/Word Parser. The concordance is based on the contents of the Input Text.

content description?
A stream of ASCII characters.

name: **Output Filename**

alias: none

where used?
Passed from the User Command Parser to the Output Formatter.

how used?
Used to specify which file to open/create to receive the Output Text.

content description?
A string of characters.

name: **Output Text**
alias: none

where used?
1.) Passed from the Output Formatter to the Output Text Document

how used?
Used to represent the contents of the Concordance Structure.

content description?
Lines of text, delimited by carriage returns, containing the Words and Coordinates of Appearance of the Words

name: **Status**
alias: none

where used?

Passed from the Skip-Word Module to the Line/Word Parser Module.

how used?
Used to specify if a given word is a member of the Skip-Word List.

content description?
Boolean

name: **User Command Line**
alias: UCL

where used?
Passed from the Console to the User Command Parser.

how used?
Used to invoke the Document Concordance Generator and to specify an Input Filename and Output Filename

content description?
DCG <Input Filename> <Output Filename>

supplementary information?
The elements within the UCL must be delimited by a blank space.

name: **Words**
alias: none

where used?
1.) Passed from the Line/Word Parser to the Skip-Word Module.
2.) Passed from the Line/Word Parser to the Structure Builder.
3.) Passed from the Structure Builder to the Concordance Structure.
4.) Retrieved from the Concordance Structure by the Output Formatter.

how used?
Used to specify the words of the concordance.

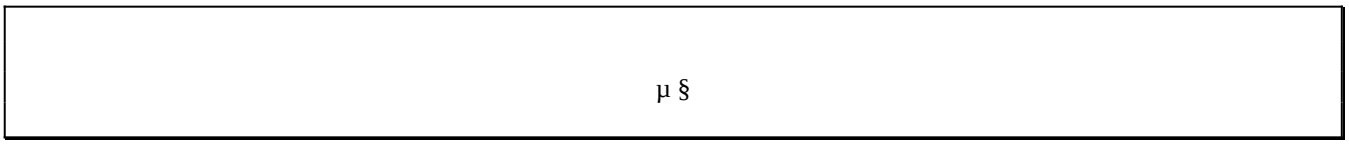
content description?
A string of characters delimited by white space.

3.2 Derived Program Structure

DCG Calling Chain

µ §

User Command Parser:



µ §

Line/Word Parser:

µ §

Skip Word Module:

µ §

Structure Builder:

µ §

Output Formatter:

µ §

3.3 Interfaces Within Structure

Concordance Structure

The Concordance Structure is used in an intermediate fashion to hold the master list of words that have been found in the Input Text Document. The Concordance Structure is also responsible for holding the lists of Coordinates of Appearance that have been determined for each word.

The so-called "spine" of the Concordance Structure consists primarily of a doubly-linked-list, as described on pages 71-77 of the CS Parts Document. Each element of this list contains a word, used as a key, as well as a pointer to a singly-linked-list, which are known as the "rib" lists. Each "rib" list maintains the Coordinates of Appearance for that word. This singly-linked- list is described on pages 83-94 of the CS Parts Document. For an abstract representation of the Concordance Structure, please refer to figure 3.3.

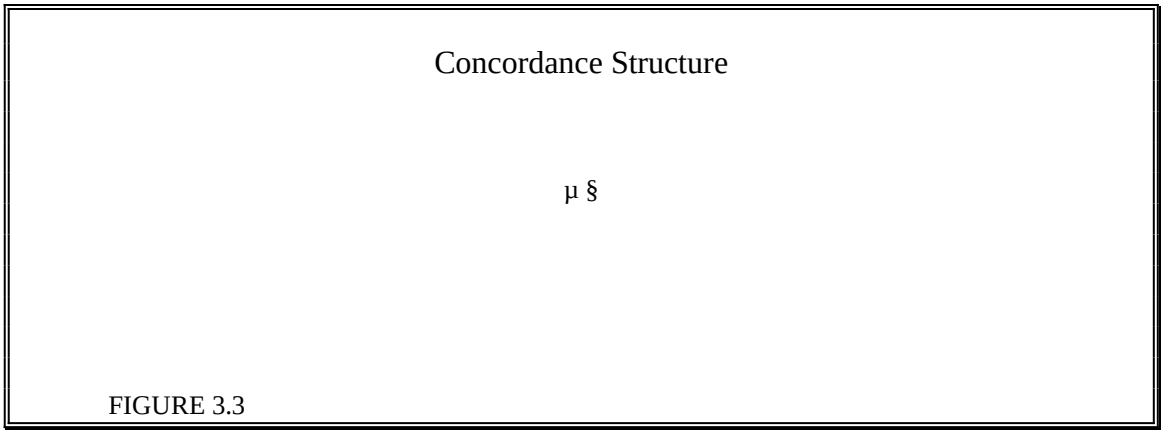


FIGURE 3.3

Skip Word List:

The SkipWord list is to be implemented using a singly-linked list as described on pages 83-94 of the CS Parts Manual. This list is created by the User Command Parser, once it has been determined that the Skip.List file exists, can be opened and read. Each link of this list contains a single word from the Skip.List file. For an abstract representation of the SkipWord List, please refer to figure 3.4.

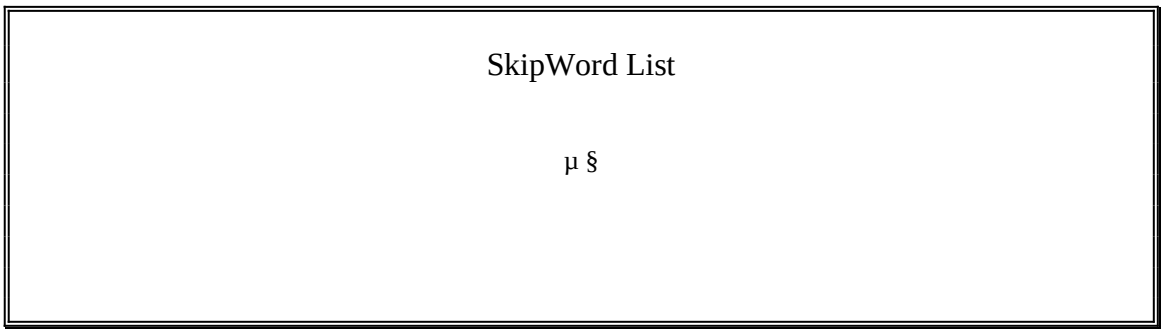


FIGURE 3.4.

ErrorMessages:

The following are the error messages used by various modules of the Document Concordance Generator.

ERROR - Unable to open specified input file "<InputFileName>".

ERROR - Unable to read from input file "<InputFileName>".

ERROR - Specified output file "<OutputFileName>" already exists. Overwrite? (Y/N)

ERROR - Unable to write to "<OutputFileName>".

ERROR - Correct syntax is: DCG InputFileName OutputFileName

ERROR - Unable to open Skip.List file.

ERROR - Unable to read from Skip.List file.

4. Module Design

User Command Parser

Processing Narrative:

The UCP operates on the command line to determine the desired input and output file names, it also creates the SkipWord List.

Upon invocation of the DCG, the UCP is called to parse the command line into its constituent elements which are essentially filenames. The command line is checked to determine whether it contains three elements: the program name, the input filename, and the output filename. If not, an error has occurred, and a message is displayed with the correct syntax for input to the DCG. An attempt to open the Input Text Document is then made. If the UCP is unable to open the file, an error has occurred and a message is displayed to that effect. If the file is successfully opened, its FILE_TYPE is passed to the Line/Word Parser. An attempt to open the Output Text File is made next. If the UCP is able to open the file, then the user is prompted that this file already exists and is asked whether it should be over-written. If the user responds negatively, the DCG then terminates normally. If unable to open the file, a new file is created. The FILE_TYPE of the Output Text Document is passed to the Output Formatter.

In addition, the UCP attempts to open a file used to create the SkipWord List. The name of this file is Skip.List. If unable to open this file, the DCG displays an error message and terminates with an error. If the file exists and can be opened, the UCP repeatedly gets a new line from the file, each of which contains a skip-word. As each word is found, a node is added to the SkipWord List.

Interface Description:

procedure UserCommandParser(UserCommandLine: in STRING)

Design Language Description:

```
procedure UserCommandParser(UserCommandLine: in STRING) is

begin

        (* initialize the LINE_PARSE package *)
```

```
        Initialize(UserCommandLine);

        (* check for correct number of parameters *)
        if ARGC < 3 then
                raise NotEnoughParameters;
        end;
        if ARGC > 3 then
                raise TooManyParameters;
        end;

        (* get the filenames *)
        InputFileName := ARGV(1);
        OutputFileName := ARGV(2);

        (* try to open the input file *)
        OPEN(InputFileType,InputFileName,Status);
        if (Status = NOT_OK) then
                raise InputFileNotOK;
                terminate;
        end;

        (* try to open the output file *)
        OPEN(OutputFileType,OutputFileName,Status);
        if Status = NOT_OK then
                raise OutputFileNotOK;
                terminate;
        end;

        (* try to open the skip word file *)
        OPEN(SkipWordFileType,"Skip.List",Status);
        if (Status = NOT_OK) THEN
                raise SkipFileNotOK;
                terminate;
        end;

        (* create the skipword list *)
        Initialize(SkipWordList);
        while (NOT EOF) do
                GetLine(SkipWordFileType,NewLine);
                GetWord(NewLine,SkipWord)
                AddNode(SkipWordList,SkipWord);
        end;
        SkipWord.Init(SkipWordList);

        (* init the structure builder module *)
        Initialize(ConcordanceStructure);
        StructureBuilder.Init(ConcordanceStructure,
                                        InputFileName,OutputFileName
                                        OutputFileType);

end UserCommandParser;
```

Packages Used

Package Name	Description
LineWordParser	(DCG) Parses the Input Text Document

SkipWord Module	(DCG) Used in Initialization
StructureBuilder	(DCG) Used in Initialization
CommandLineInterface	(CS Parts) Retrieves the command line
InputFile	(CS Parts) Opens the Input Text Document
OutputFile	(CS Parts) Opens the Output Text Document
Console	(CS Parts) Outputs messages to the console

Data Organization:

```
UserCommandLine:        STRING;
InputFileName:      STRING;
OutputFileName:     STRING;
NewLine:                STRING;
SkipWord:               STRING;

InputFileType:      FILE_TYPE;
OutputFileType:     FILE_TYPE;
SkipWordFileType:       FILE_TYPE;

ConcordanceStructure:   DOUBLY_LINKED_LIST;
SkipWordList:       SINGLY_LINKED_LIST;
```

Line/Word Parser

Processing Narrative:

The Line/Word Parser functions to sequentially scan the Input Text Document and break it into is constituent words.

As each new line is gotten from the Input Text Document, it is scanned in a character-by character fashion. The CS Parts Package STRING_SCANNER is used in this capacity. As each word is identified, it is passed to the SkipWord Module to determine its status.

Any word appearing on the end of a line with a hyphen, shall be treated as incomplete, and shall be held over until the next line is fetched. The CoordinatesOfAppearance of these words shall reflect the placement fo the second half of these words.

In addition, the number of lines gotten from the Input Text Document is continually updated for the purposes of determining which page the word came from. Each page consists of 60 lines.

Once a Word and its CoordinatesOfAppearance have been determined, they are passed to the Structure Builder.

Interface Description:

```
procedure LineWordParser(InputFileType: in FILE_TYPE)
```

Design Language Description:

```
procedure LineWordParser(InputFileType: in FILE_TYPE) is

begin

        (* get each line from the input file *)
        while (NOT EOF) do
                GetLine(InputFileType,NewLine);

                (* keep track of lines, pages *)
                INC(LineCount);
                if (LineCount = 61) then
                        INC(PageCount);
                        LineCount := 1;
                end;

                theScanner := MakeScanner(NewLine);

                (* get each word from the line *)
                while (More(theScanner)) do

                        ScanWord(theScanner,WordFound,
                                        theWord,DoNotSkip);

                        (* check if the scanner has found a word *)
                        if (WordFound) then
                                Status := SkipWord(theWord);

                                (* if word ok pass to SB *)
                                if (Status) then
                                        StructureBuilder(theWord,
                                                LineCount,PageCount);
                                end;
                        end;
                end;
        end;

end LineWordParser;
```

Packages Used:

Package Name	Description
SkipWordModule	(DCG) Determines SkipWord Status
StringScanner	(CS Parts) Scans the lines for words
StructureBuilder	(DCG) Builds the ConcordanceStructure

InputFile	(CS Parts) Gets lines from the Input Text Document

Data Organization:

```
InputFileType:   FILE_TYPE;
NewLine:             STRING;
LineCount:           INTEGER;
PageCount:           INTEGER;
theScanner:          STRING_SCANNER;
WordFound:           BOOLEAN;
theWord:             STRING;
DoNotSkip:           BOOLEAN := FALSE;
Status:          BOOLEAN;
```

Skip-Word Module

Processing Narrative:

The SkipWord Module functions to determine whether a give word is a member of the SkipWord List. The SkipWord Module sequentially scans the SkipWord list to see whether the word passed in matches any on the list. If a match is made, TRUE is returned to the caller. Otherwise, FALSE is returned.

Interface Description:

```
function SkipWord(WordToCheck: in STRING) return BOOLEAN
```

Design Language Description:

```
function SkipWord(WordToCheck: in STRING) return BOOLEAN is

begin

        (* find the head of the list *)
        First(SkipWordList);

        while (Not Last(SkipWordList)) do
                if CurrentElement.Word = WordToCheck then
                        return TRUE;
                end;

                Next(SkipWordList);
        end;

        return FALSE;

end SkipWord;
```

Packages Used:

Package Name	Description
Singly_Linked_List	(CS Parts) Manages the SkipWord List

Data Organization:

Not Applicable.

Structure Builder

Processing Narrative:

The SB functions to create and manage the Concordance Structure. As each WordRecord is passed to the SB, the Word element of the WordRecord is used as a key to search the Concordance Structure. If the Word has already occurred within the document, then Coordinates of Appearance element of the WordRecord is added to the end of the 'rib' list.

If the Word has not yet appeared within the input file, then the Word is added at the appropriate place within the list, and the Coordinates of Appearance are recorded as well. If any errors occur during this process, an error message will be displayed on the screen, and the function shall return FALSE. Otherwise, this module shall return TRUE.

Interface Description

```
function StructureBuilder(Word: STRING;
                          PageNumber: INTEGER;
                          LineNumber:  INTEGER) return BOOLEAN
```

Design Language Description:

```
function StructureBuilder(Word: STRING;
                          PageNumber: INTEGER;
                          LineNumber:  INTEGER) return BOOLEAN is

begin

    (* CurrentElement points to head of list *)
    CurrentElement := ConcordanceStructure.First;

    (* find the insertion point for the word *)
    while (Word >= CurrentElement.Word) do
        CurrentElement := ConcordanceStructure.Next;
    end;

    (* if the word is already in the list *)
    if (CurrentElement.Word = Word) then

        (* find the head of the list *)
        PlaceElement := CurrentElement.RibList.First;

        (* insert the coords of appearance *)
        while (PlaceElement != CurrentElement.RibList.Last) do
            PlaceElement := PlaceElement.Next;
        end;

        (* add a node for the coordinates of appearance *)
        AddNode(PlaceElement);
        PlaceElement.Page := PageNumber;
        PlaceElement.Line := LineNumber;
        return TRUE;

    (* otherwise add the element *)
    else
```

```
                AddNode(CurrentElement);
                CurrentElement.Word := Word;
                PlaceElement := CurrentElement.RibList.First;
                PlaceElement.Page := PageNumber;
                PlaceElement.Line := LineNumber;
                return TRUE;
        end;

        return FALSE;

end StructureBuilder;
```

Packages Used:

Package Name	Description
Lists	(CS Parts) Manages the ConcordanceStructure "ribs".
Case_Insensitive_String_ Comparison	(CS Parts) Used to compare words to those already in the list.
Singly_Linked_List	(CS Parts) Manages the ConcordanceStructure "spine".

Data Organization:

```
CurrentElement:  ELEMENT_OBJECT;
PlaceElement:    ELEMENT_OBJECT;
```

Output Formatter

Processing Narrative:

The OF operates on the Concordance Structure to build the Output Text Document. This is accomplished by a sequential scan of the Concordance Structure. As each link of the Concordance Structure 'spine' list is traversed, the word occurring in that link is printed to the Output Text Document. Next, the 'rib' list of each link is scanned in a sequential fashion. The OF prints each coordinate of appearance as they appear in the 'rib' list. When all the Coordinates of Appearance have been printed for that word, the OF moves on to the next link in the 'spine' list. This is repeated until every word in the Concordance Structure and its Coordinates of Appearance have been printed to the file.

Interface Description

```
procedure OutputFormat(ConcordanceStructure: in LIST_TYPE)
```

Design Language Description

```
procedure OutputFormat(ConcordanceStructure: in LIST_TYPE) is

begin

        (* find the head of the list *)
        CurrentElement := ConcordanceStructure.First;
```

```
        (* print out the title page *)
        PrintTitlePage();

        (* print out all the words and their coordinates *)
        while CurrentElement != ConcordanceStructure.Last do

                Print(CurrentElement.Word);
                NewLine();
                PlaceElement := CurrentElement.RibList.First;

                while PlaceElement != CurrentElement.RibList.Last do
                        Print(PlaceElement.PageNumber);
                        PrintTab();
                        Print(PlaceElement.LineNumber);
                        NewLine();
                        PlaceElement := PlaceElement.Next;
                end;
                NewLine();

                CurrentElement := CurrentElement.Next;

        end;

end OutputFormat;
```

Packages Used:

Package Name	Description
Doubly_Linked_List	(CS Parts) Manages the ConcordanceStructure
Singly_Linked_List	(CS Parts) Manages the ConcordanceStructure
FOF	(CS Parts) Formats the Output Text Document

Data Organization

```
CurrentElement:  ELEMENT_OBJECT;
PlaceElement:    ELEMENT_OBJECT;
```

5. File Structures and Global Data

5.1 External File Strucutres

Input Text Document

Logical Structure - The logical structure of the Input Text Document is that of an ordinary text file. Individual lines are delimited by carriage returns, and individual words are delimited by blank space, commas, and other punctuation marks. Please refer to figure 5.1 for the logical structure of the Input Text Document.

Logical Record Description - Not Applicable.

Access Method - Sequential

This course is the laboratory component of
the ECE 493 Software Engineering course.
During the laboratory course the student will
apply formal software engineering techniques to
the development of a software product. Maximum
re-use of software components will be empha-
sized to ensure timely development of a sig-
nificant product in a short period of time.

Figure 5.1

Skip Word File

Logical Structure - The logical structure of the SkipWord file is straightforward. All the words to be omitted from the concordance appear in a column, with one word per line. Please refer to figure 5.2 for the logical structure of the SkipWord File.

Logical Record Description - Not Applicable.

Access Method - Sequential.

a
an
the
and
or
then
else
of
but
is
are
not
to
that
from
it
itself
in
out
very
most
it's
also

Figure 5.2

Output Text Document

Logical Structure - The logical structure of the Output Text Document may be seen in figure 5.3.

Logical Record Description - Not Applicable.

Access Method - Sequential.

apply
line: 4 page: 1

be
line: 6 page: 1

component
line:1 page: 1

components
line: 6 page: 1

course
line: 1 page: 1
line: 2 page: 1

development
line: 5 page: 1
line: 7 page: 1

during
line:3 page: 1

ece
line: 2 page: 1

empha-sized
line: 7 page: 1

Figure 5.3

5.2 Global Data

None.

5.3 File and Data Cross Reference

Input Text Document:

Words: Please refer to figure 5.4. In this figure, the items appearing in boldface are considered words for the

purposes of the concordance.

This course is the **laboratory component** of the **ECE 493 Software Engineering course**. **During** the **laboratory course** the **student will apply formal software engineering techniques** to the **development** of a **software product**. **Maximum re-use** of **software components will be emphasized** to **ensure timely development** of a **significant product** in a **short period** of **time**.

Figure 5.4

SkipWord List:

SkipWords - Please refer to figure 5.2. Each sequence of characters on each line is considered to be a skip word.

Output Text Document:

Words: Please refer to figure 5.5. Each sequence of characters in boldface is considered to be a word for the purposes of the concordance.

CoordinatesOfAppearance: Please refer to figure 5.5. Each number in italics is considered to be a coordinate of appearance.

apply
line: *4* page: *1*

be
line: *6* page: *1*

component
line:*1* page: *1*

components
line: *6* page: *1*

Figure 5.5

6. Requirements Cross Reference

For a complete cross reference between the elements of this document and the corresponding Software Requirements Specification, please refer to table 6.1.

Item	Requirements Spec. Location
User Command Parser	Section 3.1
Line/Word Parser	Section 3.1
Skip Word Module	Section 3.1
Structure Builder	Section 3.1
Output Formatter	Section 3.1
SkipWord List	n/a
Concordance Structure	Section 2.2
Input File	Section 2.2
Skip Word File	n/a
Output File	Section 2.2
Testing Strategy	Section 7.1

Table 6.1

7. Test Provisions

7.1 Test Guidelines

Table 7.1 is a further refinement of the test plan information as presented in the Document Concordance Generator Software Requirements Specification. The final revision of the overall test plan shall be presented in the Document Concordance Generator Software Test Plan.

Test Class	Test Description
User Command Parser Tests	Echo Command Line
	Parameter Count
	Open Input Text File
	Open Output Text File

	Open Skip Word File
Line/Word Parser Tests	Echo Line
	Echo Word
	Hyphenation Recognition
	Punctuation Recognition
	Line Count
	Page Count
Skip-Word Module Tests	Identification of Skip Words
Structure Builder Tests	Incremental Structure Printout
	Maximum Size
Output Formatter Tests	Title Page
	Output Content
	Page Layout

Table 7.1

7.2 Integration Strategy

The overall integration strategy is incremental in nature, and the group has identified five distinct phases of module integration. Since the nature of the DCG is rather sequential, the various modules shall be integrated as shown in table 7.2.

Phase	Modules Involved	Tests Planned
1	UCP	User Command Parser Tests
2	UCP, LWP	Line/Word Parser Tests
3	UCP, LWP, SWM	Skip-Word Module Tests

4	UCP, LWP, SWM, SB	Structure Builder Tests
5	UCP, LWP, SWM, SB, OF	Output Formatter Tests

Table 7.2

7.3 Special Considerations

None.

8. Packaging

No special packaging requirements have been identified for the DCG.

8.1 Special Program Overlay Provisions

No special program overlay provisions have been identified for the DCG.

8.2 Transfer Considerations

No transfer considerations have been identified for the DCG.

9. Special Notes

None.

10. Appendices

None.