1. Scope of Testing

A strategy for testing integrates software test case design techniques into a well-planned series of steps that result in the successful construction of software. As importantly, a software testing strategy provides a road map for the software developer, the quality assurance organization, and the customer -- a road map that describes the steps to be conducted as part of testing, when these steps are planned and then undertaken, and how much effort, time, and resources will be required. Therefore, any testing strategy must incorporate test planning, test case design, test execution and the resultant data collection and evaluation.

- R. Pressman.

2. Test Plan

The overall test plan build is incremental in nature and the group has identified 6 distinct phases of module testing. Since the nature of the DCG is rather sequential, the various modules will be tested as shown in table 2.1.

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
6	DCG	Black Box Tests

Table 2.1

2.2 Schedule

The start of each module testing begins at the completion of each module, during the writing of the DCG. Upon completion of the module testing the Black Box testing begins. The testing is completed only at the end of the project.

2.3 Overhead Software

The groups unfamiliarity with test benches, stubs, and drivers for the ada compiler leads us to use only the ada compiler's strong type checking and the extensive use of print statements to trace the progress of the component functionality.

2.4 Environment and Resources

The environment for testing is the Unix based Sun Sparc workstations with the ada compiler. Resources include the past documents Generated by the group in the Engineering process of the DCG.

3. Test Procedure: Build *n*

3.1 Order of Integration

The overall test plan integration is incremental in nature and the group has identified 6 distinct phases of module integration. Since the nature of the DCG is rather sequential, the various modules will be integrated as shown in table 3.1.

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
6	DCG	Black Box Tests

Table 3.1

3.1.1 Purpose

The purpose of the integration strategy is to insure each phase of the software development is working correctly before the next step is undertaken. This is an attempt to reduce the complexity of errors found and thus reducing the overall length of time it takes to produce the completed software package.

3.1.2 Modules to be Tested

The modules to test are the User Command Parser, the Line/Word Parser, the Skip Word module, the Structure Builder module, and the Output Formatter Module.

These modules would all be included in the test plan if it were to be completed; however, this document is just being expanded for a few modules to demonstrate the use of the test plan.

3.2 Unit Tests For Modules in Build

Test Class	**Test Description**
User Command Parser Tests	Echo Command Line
	Parameter Count
	Echo Usage Message
	Check Existing Output Filename
	Open Text File
	Open Skip Word File
	Echo Skip Words
Line/Word Parser	Echo Line
	Echo Word
	Hyphenation Recognition
	Punctuation Recognition
	Line Count
	Word Count
Skip Word Module Tests	Identification of Skip Words

	Echo Return Value
Structure Builder Tests	Incremental Structure Printout
	Maximum Size
Output Formatter Tests	Title Page
	Output Contents
	Page Layout
Black Box Tests	Compare to Hand Generated Concordances
	"Test1.txt"
	"Test2.txt"
	"Test*.txt"

Table 3.2

3.2.1 Description of Tests for Module *m*

Phase 1:

Test Class	**Test Description**
User Command Parser Tests	Echo Command Line
	Parameter Count
	Echo Usage Message
	Check Existing Output Filename
	Open Text File
	Open Skip Word File
	Echo Skip Words

Table 3.2.1.1

The tests listed in table 3.2.1.1 are the tests to be run on this module. The echo command line is used to test the ArgV function to insure the commands can be taken from the command line to be used for opening files. The echo parameter count is used to check the number of arguments taken from the command these numbers are used to output usage messages when the command line is incorrectly used. Echo usage message is tested by putting in an incorrect number of arguments on the command line to see if the correct usage message is output or that no message is output when the number of arguments are correct. Check for existing Output file name this is used to see if the output file name received already exists in the director the program resides in to prevent over writing an existing file. Open text file can be tested with the open skip word file since both files are text type. Echo skip words is the next step in testing if a file can be opened in that if the skip words from the skip word file are echoed to the screen then the file must be open.

Phase 2:

Test Class	**Test Description**
Line/Word Parser	Echo Line
	Echo Word
	Hyphenation Recognition
	Punctuation Recognition
	Line Count
	Word Count

Table 3.2.1.2

This module was not expanded on since the test document is for demonstration purposes.

Phase 3:

Test Class	**Test Description**
Skip Word Module Tests	Identification of Skip Words
	Echo Return Value

Table 3.2.1.3

This module has only two simple tests. First, to see if words passed to the module that are on the list return a True (skip it) or words not on the list return a False (don't skip it). The next step is to see if the correct return value is received by the calling function this can be done by printing the word and value immediately upon returning to the calling function.

Phase 4:

Test Class	**Test Description**
Structure Builder Tests	Incremental Structure Printout
	Maximum Size

Table 3.2.1.4

This module was not expanded on since the test document is for demonstration purposes.

Phase 5:

Test Class	**Test Description**
Output Formatter Tests	Title Page
	Output Contents
	Page Layout

Table 3.2.1.5

This module was not expanded on since the test document is for demonstration purposes.

Phase 6:

Test Class	**Test Description**
Black Box Tests	Compare to Hand Generated Concordances
	"Test1.txt"
	"Test2.txt"
	"Test*.txt"

Table 3.2.1.6

The Black Box testing begins upon the completion of the earlier integration steps. By taking known text files and passing them to the DCG then comparing them to the expected hand generated concordance is the final step in testing. Any errors found at this point should be minor; however, if errors are found and corrected the integration phase in which the module corrected was integrated should be the location unit testing is resumed at (i.e. if a problem was found in the skip module testing would resume in phase 3 and continue through each additional phase.

3.2.2 Overhead Software Description

The groups unfamiliarity with test benches, stubs, and drivers for the ada compiler leads us to use only the ada compiler's strong type checking and the extensive use of print statements to trace the progress of the component functionality.

3.2.3 Expected Results

Incorporated into section 3.4.

3.3 Test Environment

The environment for testing is the Unix based Sun Sparc workstations with the ada compiler.

3.3.1 Special Tools or Techniques

Use of the ada compiler's strong type checking and extensive use of print statements to trace the progress of the component functionality are the only techniques used in the module testing.

3.3.2 Overhead Software Description

The groups unfamiliarity with test benches, stubs, and drivers for the ada compiler leads us to use only the ada compiler's strong type checking and the extensive use of print statements to trace the progress of the component functionality.

3.4 Test Case Data

Phase 1:

Test Class	**Test Description**
User Command Parser Tests	Echo Command Line
	Parameter Count
	Echo Usage Message
	Check Existing Output Filename
	Open Text File

	Open Skip Word File
	Echo Skip Words

Table 3.4.1

The tests for the user command parser are listed in table 4.1. For the echo command line test the DCG is initiated with an input file name and an output file name which are echoed to the screen. For the parameter count test various numbers of arguments are placed on the command line and the number of parameters are echoed to the screen. For the echo usage message test various numbers of parameters are entered on the command line and fewer then 3 produces a not enough parameters message along with the usage, 3 produces an echo of the input and output file names, and more than 3 produces a too many parameters message along with the usage message. To check for the existence of an output file put a filename in the output file parameter location that currently exists in the current directory and a file already exists message should be raised; if a new file name is used no message should be echoed to the screen. The open skip file and echo skip words tests can be incorporated together since a good test if a file has been opened is to print out it's contents to the screen. The skip file is input and when the dcg is invoked then the words pulled from the skip list file are printed to the screen.

Sample input and output echo command line test:

```
dcg  Infile.txt  Outfile.txt                The input file is   Infile.txt
                                            The output file is Outfile.txt
```

Sample input and output for the parameter count:

```
dcg  Infile.txt  Outfile.txt                The parameter count is 3
dcg  Infile.txt                             The parameter count is 2
dcg  Infile.txt  Skipfile.txt Outfile.txt   The parameter count is 4
```

Sample input and output for the usage message:

```
dcg  Infile.txt  Outfile.txt                The input file is   Infile.txt
                                            The output file is Outfile.txt

dcg  Infile.txt                             Not enough parameters.
                                            usage:  dcg   <InputFileName>  <OutputFileName>

dcg  Infile.txt  Skipfile.txt Outfile.txt   Too many parameters.
                                            usage:  dcg   <InputFileName>  <OutputFileName>
```

dcg	Not enough parameters.
	usage: dcg <InputFileName> <OutputFileName>

Sample input and output for checking for existence of Output file:

dcg Infile.txt Outfile.txt	Output file name is valid.
dcg Infile.txt Infile.txt	The output file already exists.

Sample input skip word list:

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

Echoed output to screen from skip word list:

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

Test Class	**Test Description**	**Test O.K.**
User Command Parser Tests	Echo Command Line	
	Parameter Count	
	Echo Usage Message	
	Check Existing Output Filename	

	Open Text File
	Open Skip Word File
	Echo Skip Words

Table 4.1

Phase 2:

Test Class	**Test Description**
Line/Word Parser	Echo Line
	Echo Word
	Hyphenation Recognition
	Punctuation Recognition
	Line Count
	Word Count

Table 3.4.2

This module was not expanded on since the test document is for demonstration purposes.

Test Class	**Test Description**	**Test O.K.**
Line/Word Parser	Echo Line	
	Echo Word	
	Hyphenation Recognition	
	Punctuation Recognition	
	Line Count	
	Word Count	

Table 4.2

Phase 3:

Test Class	**Test Description**
Skip Word Module Tests	Identification of Skip Words
	Echo Return Value

Table 3.4.3

For tests on the skip word module every word contained in the skip word list should be passed to the module and the value returned for each should be true. Then for any words not contained in the list the value of false should be returned.

Sample input words from the standard skip list file:

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

Return value to be printed out :

True - the word was skipped

Return value for other words to be printed out:

False - the word was not skipped

Test Class	**Test Description**	**Test O.K.**
Skip Word Module Tests	Identification of Skip Words	
	Echo Return Value	

Table 4.3

Phase 4:

Test Class	**Test Description**
Structure Builder Tests	Incremental Structure Printout
	Maximum Size

Table 3.4.4

This module was not expanded on since the test document is for demonstration purposes.

Test Class	**Test Description**	**Test O.K.**
Structure Builder Tests	Incremental Structure Printout	
	Maximum Size	

Table 3.4

Phase 5:

Test Class	**Test Description**
Output Formatter Tests	Title Page
	Output Contents
	Page Layout

Table 3.4.5

This module was not expanded on since the test document is for demonstration purposes.

Test Class	**Test Description**	**Test O.K.**
Output Formatter Tests	Title Page	
	Output Contents	
	Page Layout	

Table 4.5

Phase 6:

Test Class	**Test Description**
Black Box Tests	Compare to Hand Generated Concordances

	"Test1.txt"
	"Test2.txt"
	"Test*.txt"

Table 3.4.6

Tests include input files with known concordance structures to test the functionality of the modules in files named Test*.txt and the known outputs are in Test*.out files. The DCG should be tested for an input file with standard text, a text file with only words from the standard skip word list, a file with multiple pages of just a single word, and a multiple page document where each page is the same to determine if the line page functions work correctly.

Sample input text file:

A strategy for testing integrates software test case design techniques into a well-planned series of steps that
result in the successful construction of software. As importantly, a software testing strategy provides a road map for
the software developer, the quality assurance organization, and the customer -- a road map that describes the steps
to be conducted as part of testing, when these steps are planned and then undertaken, and how much effort, time, and
resources will be required. Therefore, any testing strategy must incorporate test planning, test case design, test
execution and the resultant data collection and evaluation.

Expected output file:

```
any
        line: 6    page:4
assurance
        line: 3    page: 1
case
        line: 1    page: 1
        line: 6    page: 1
collected
        line: 7    page: 1
conducted
        line: 4    page: 1
constructed
        line: 2    page: 1
customer
        line: 4    page:1
        ...
undertaken
        line: 5    page: 1
```

Sample input text file:

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

Expected output file:

```
All the words in the input file were contained in the skip word list.
```

Sample input text file:

```
hello          hello          hello          hello          hello
hello          hello          hello          hello          hello
hello          hello          hello          hello          hello
hello          hello          hello          hello          hello
                         ...
hello          hello          hello          hello          hello
```

Expected output file:

```
hello
     line: 1    page: 1
     line: 2    page: 1
     line: 3    page: 1
     line: 4    page: 1
     line: 5    page: 1
                ...
     line: 59   page: 3
```

Test Class	**Test Description**	**Test O.K.**
Black Box Tests	Compare to Hand Generated Concordances	
	"Test1.txt"	
	"Test2.txt"	
	"Test*.txt"	

Table 4.6

3.5 Expected Results for Build *n*

Incorporated into section 3.4.

4. Actual Test Results

Incorporated into section 3.4.

5. References

Document Concordance Generator - Design Specification
by: Eric Brickner and Chris Blanchard

Software Engineering - A Practitioner's Approach
by: Roger S. Pressman

Software Test Plan - Outline and Requirements
by: Professor Carter

6. Appendices

None.