

OREGON STATE

UNIVERSITY

COMPUTER

SCIENCE

DEPARTMENT

VIGRAM
A Program Understanding and Complexity Metric Analysis tool for Pascal Programs.

Kritawan Kruatrachue
Dr. Ted G. Lewis
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

88-60-16

VIGRAM

*A Program Understanding and Complexity
Metric Analysis tool for Pascal Programs.*

by

Kritawan Kruatrachue

A Research Paper Submitted to Partially Fulfil the
Requirements for the Degree of Master of Science

Major Professor : Dr. Ted G. Lewis

Department of Computer Science

Oregon State University

Corvallis, OR 97331

July, 1988

Acknowledgements

I am deeply grateful to my advisor, Professor Ted Lewis, for his help, guidance, understanding and encouragement. Also, I would like to thank my teachers : Professor Timothy A. Budd, Professor Bruce D. D'Ambrosio, Professor Toshimi Minoura, Professor Charles Swart who taught and guided me through out my Master's program.

I thank my friend Thit Siriboon, Sherry Yang and Visith Chavasit for their help and suggestion.

I would like to express my gratitude to my parent and my aunts for their love, support and encouragement throughout my life. Finally, I thank my brothers for all their love and support.

TABLE OF CONTENTS

ABSTRACT.....	1
1. Introduction.....	2
1.1 O.S.U. Project.....	2
1.2 VIGRAM and O.S.U.	4
2. Background : Plum Diagram Editor.....	5
3. VIGRAM.....	11
3.1 What is VIGRAM ?.....	11
3.2 Why VIGRAM ?.....	11
3.2.1 Program understanding.....	11
3.2.2 Complexity Analysis.....	12
3.3 VIGRAM's Three Main Functions.....	12
3.4 Program Understanding	13
3.5 Complexity Metric Analysis	15
3.5.1 Weight of Difficulty	15
3.5.2 Halstead complexity metrics.....	16
3.5.3 Berry-Meekings program characteristics.....	17
3.5.4 Other counts.....	17

4. Using VIGRAM.....	18
4.1 Menu Reference.....	18
4.2 Tutorial.....	21
5 Implementation.....	32
5.1 Data Structure.....	32
5.2 Possible Extensions.....	34
5.3 Requirements and limitations.....	34
5.4 Application Statistics.....	35
6. Summary.....	36
6.1 Program Understanding.....	36
6.2 Complexity Metrics.....	36
7. Bibliography.....	40

TABLE OF FIGURES

Figure 2.1	The Plum Diagram Editor Visual Building Blocks (Plums). From left to right, these plums represent constant definition, type definition, variable declaration, assignment, if, repeat, while, case, for, with, Macintosh toolbox routine, procedure or function call, read(ln) or write(ln), and ExitToShell Macintosh toolbox routine.....	5
Figure 2.2a	A piece of Pascal code.....	6
Figure 2.2b	The plum diagram representing the piece of Pascal code in Figure 2.2a.....	6
Figure 2.3	Double mouse-clicking on a plum reveals its hidden detailed information. The arrow-line indicates the first plum in Figure 2.2b.....	7
Figure 2.4a	A Pascal procedure.....	8
Figure 2.4b	The first level plum diagram of the Pascal procedure in Figure 2.4a.....	9
Figure 2.5	Double mouse-clicking on a plum's "next level" icon reveals the hidden next level. The arrow-lines indicate the next levels of the while-loop plum diagram in Figure 2.4b.....	9

Figure 3.1	Hierarchical slicing of the Pascal procedure in Figure 2.4a on variables : "count" and "average"....	14
Figure 4.1	Overall menu items for VIGRAM.....	18
Figure 4.2	Source file window and procedure & function list window.....	21
Figure 4.3	Selecting a procedure or function to analyze.....	22
Figure 4.4	Plum Diagram for procedure "FindAverage" in Figure 2.4a.....	23
Figure 4.5	Heading, Definition & Declaration Weight Setting Dialog.....	25
Figure 4.6	Statement Weight Setting Dialog.....	25
Figure 4.7	Constant Weight Setting Dialog.....	26
Figure 4.8	Variable Weight Setting Dialog.....	26
Figure 4.9	Operator Weight Setting Dialog.....	27
Figure 4.10	Sample complexity metric report.....	27
Figure 4.11	Description of a complexity metric report under "About Quality Metrics..." item of "apple" menu.....	28
Figure 4.12	Description of a complexity metric report under "About Quality Metrics..." item of "apple" menu.....	29
Figure 4.13	Slice-criterion dialog.....	30

Figure 4.14	A hierarchical sliced plum diagram obtained from settings shown in Figure 4.13.....	3 1
Figure 5.1	The main data structure.....	3 3
Figure 6.1	Difficulty Comparison Table.....	3 8
Figure 6.2	Total words is a factor of "Weight of Difficulty".	3 8
Figure 6.3	Scatter Graph showing relation between "Weight of Difficulty" and "Halstead's Difficulty".....	3 9

ABSTRACT

This report describes "VIGRAM" (Visual Programming) which is a program understanding and complexity metric analysis tool for Pascal programs. VIGRAM is implemented on the Macintosh as one part of the "O.S.U." (Oregon Speedcode Universe) project. With VIGRAM, the source code of a Pascal procedure can be displayed as a visual program; program slicing on variables and control & I/O statements is permitted, and Berry-Meekings, Halstead, Weight of Difficulty, McCabe complexity metrics as well as a variety of counts i.e. the number of comment words, number of comment lines in initial block, number of tabs per line, percentage of indentation tabs, number of simple type variables and number of structure type variables are computed, automatically.

1. Introduction

1.1 O.S.U. Project

O.S.U. (Oregon Speedcode Universe) is a software development environment for design, implementation, and maintenance of large software systems. It is an experimental programming system designed for Macintosh programmers who want to rapidly produce applications. The aim of O.S.U. is to increase programmer productivity by combining rapid prototyping, reusable units, program generation, and expert systems technology under an integrated environment called "speedcode universe" [YANG 88].

O.S.U. now consists of 6 main components : RezDez, Graphical Sequencer, Code Generator, Structure Chart Editor, Plum Diagram Editor and VIGRAM. The first three components are for design and implementation of a Macintosh Pascal user interface i.e. for displaying windows, menus, dialogs, alerts and icons. Structure Chart Editor is for modular design. Plum Diagram Editor and VIGRAM are for detailed design and maintenance.

The following are brief descriptions of each component of O.S.U. [YANG 88] :

RezDez : RezDez is a tool for designing Macintosh resources through direct manipulation of graphical objects. RezDez allows a designer to construct windows, menus, dialogs, alerts and icons.

Graphical Sequencer : Once the user interface resources are created from RezDez, Graphical Sequencer is used to specify the sequence of these resources. A sequence specifies the order and conditions in which to display user interface resources.

Code Generator : Code Generator generates Macintosh Pascal source code for the resources produced from RezDez and specified sequences and conditions produced from Graphical Sequencer.

Structure Chart Editor : Structure Chart Editor is a modular design tool for Pascal procedures based on hierarchical modular decomposition. Applications can be designed by combining the user interface and its sequencing with procedures defined by the Structure Chart Editor.

Plum Diagram Editor : Plum Diagram Editor (P.D. Editor) is a graphical detailed design tool for editing Pascal procedures. A programmer can design a procedure from building blocks called plums.

1.2 VIGRAM and O.S.U.

"VIGRAM" (Visual Programming) is a program understanding and complexity metric analysis tool for Pascal programs. It is integrated with Plum Diagram Editor to do detailed design and maintenance in O.S.U..

VIGRAM's aim is to reduce maintenance cost. Since maintenance cost is a major factor in the software lifecycle, reducing maintenance effort leads to increased programmer productivity [LEWI 87].

VIGRAM :

- Draws a graphical or visual version of a Pascal source code procedure.
- Parts of the visual program can be suppressed by a technique called "slicing"
- Computes : Berry-Meekings, Halstead, Weight of Difficulty, McCabe complexity metrics as well as a variety of counts i.e. the number of comment words, number of comment lines in initial block, number of tabs per line, percentage of indentation tabs, number of simple type variables and number of structure type variables.

2. Background : Plum Diagram Editor

P.D. Editor (The Plum Diagram Editor) is a graphical tool for designing a Pascal procedure from building blocks called plums. Plums are graphical elements (icons) representing the abstractions of statements in Pascal. Figure 2.1 lists the plums used in the Plum Diagram Editor.

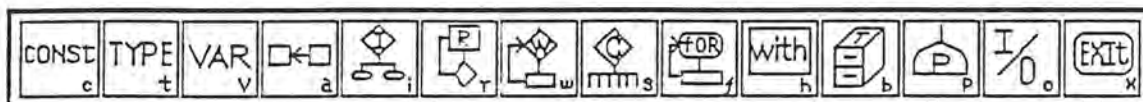


Figure 2.1 The Plum Diagram Editor Visual Building Blocks (Plums). From left to right, these plums represent constant definition, type definition, variable declaration, assignment, if, repeat, while, case, for, with, Macintosh toolbox routine, procedure or function call, read(ln) or write(ln), and ExitToShell Macintosh toolbox routine.

A new procedure is created by choosing these blocks (plums) and supplying the necessary information for each block. The plum diagram in Figure 2.2b graphically represents a piece of Pascal code drawn from Figure 2.2a.

```
Count := Count + 1;  
Sum := Sum + Number;
```

Figure 2.2a A piece of Pascal code.

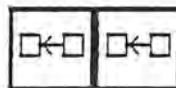


Figure 2.2b The plum diagram representing the piece of Pascal code in Figure 2.2a.

Each plum hides the detailed information by using a pictorial form to abstract a verbal form. Double mouse-clicking each plum reveals the hidden information. Figure 2.3 shows how the hidden information is revealed by double-clicking on the first plum in Figure 2.2b.

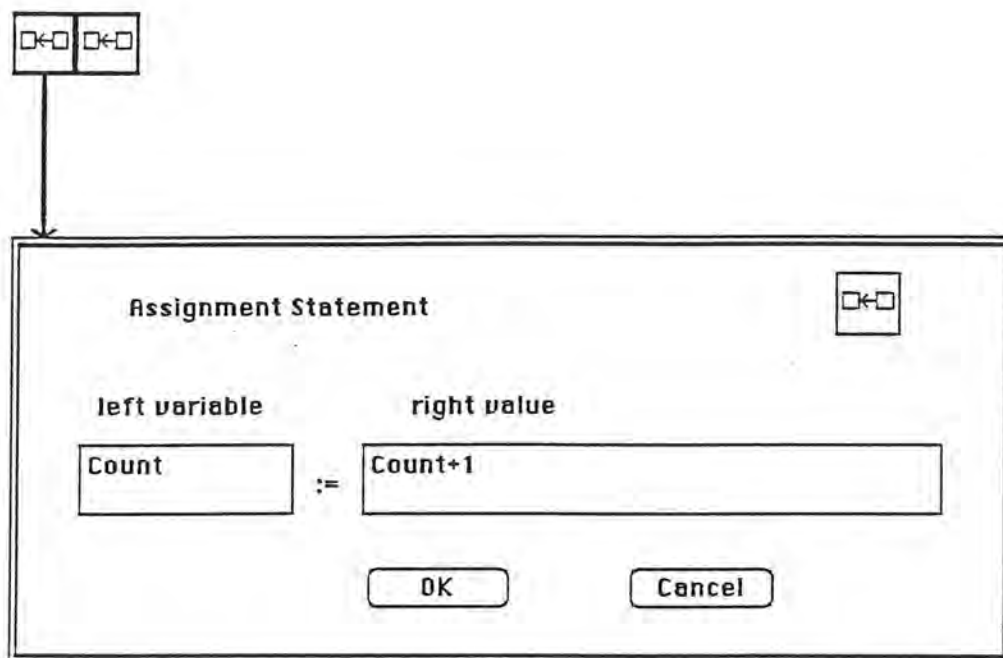


Figure 2.3 Double mouse-clicking on a plum reveals its hidden detailed information. The arrow-line indicates the first plum in Figure 2.2b.

A plum diagram also draws a source program in hierarchical manner. Each plum may have associated with it, a "next-level" plum(s) which hides the details below it in the hierarchy. Figure 2.4b shows the first level plum diagram of the Pascal procedure shown in Figure 2.4a. Figure 2.5 shows one branch of the hierarchy of Figure 2.4b.

```
PROCEDURE FindAverage;
VAR
  Count : integer;
  Sum, Average, Number : real;
  Continue : boolean;
BEGIN
  Count := 0;
  Sum := 0;
  Continue := true;
  Writeln ('Please input a number and hit return. ');
  Writeln ('To quit, enter any negative number. ');
  WHILE Continue DO
    BEGIN
      Readln (Number);
      IF Number < 0 THEN
        Continue := false
      ELSE
        BEGIN
          Count := Count + 1;
          Sum := Sum + Number;
        END;
    END;
  Average := Sum / Count;
  Writeln (Average);
END;
```

Figure 2.4a A Pascal procedure.

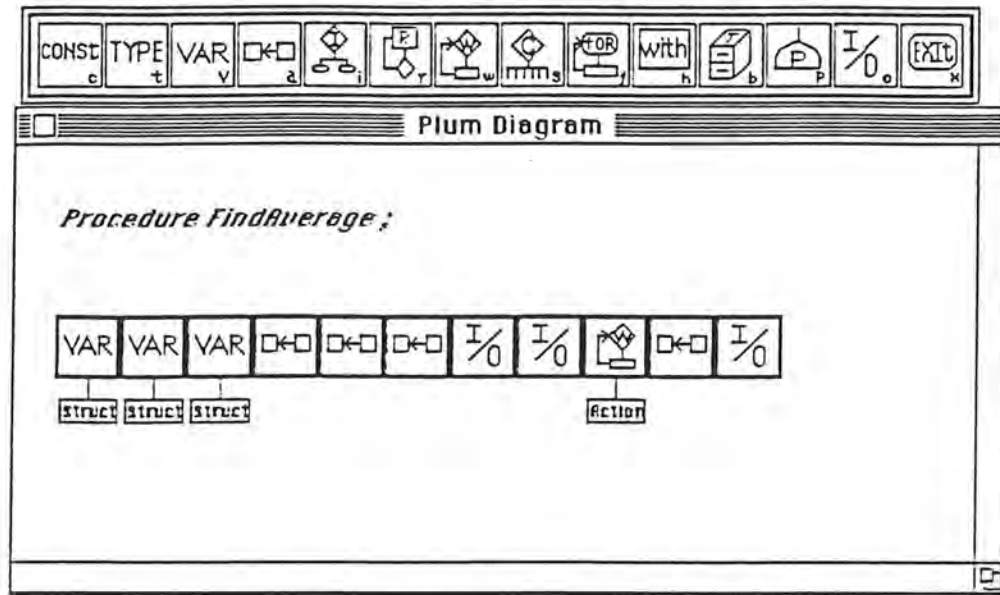


Figure 2.4b The first level plum diagram of the Pascal procedure in Figure 2.4a.

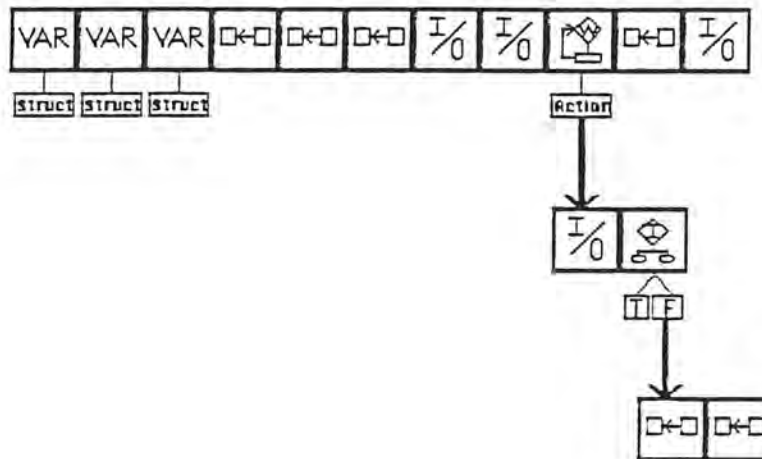


Figure 2.5 Double mouse-clicking on a plum's "next level" icon reveals the hidden next level. The arrow-lines indicate the next levels of the while-loop plum diagram in Figure 2.4b.

Once plums have been built, they can be deleted or changed. The Plum diagram can be saved as a file to be loaded and used for the next session. Also the Pascal source code representation of the plum diagram can be generated and saved.

3. VIGRAM

3.1 What is VIGRAM ?

"VIGRAM" (Visual Programming) is a complexity metric analysis and program understanding CASE (Computer-Aided Software Engineering) tool for a Pascal program. VIGRAM is one part of the "O.S.U." (Oregon Speedcode Universe) project. VIGRAM can also be a stand-alone application.

3.2 Why VIGRAM ?

3.2.1 Program understanding

One very important step in software maintenance is the task of program understanding. In the application to program understanding, VIGRAM uses graphical icons to visualize source code for these reasons.

1. Pictures are visual and multidimensional [PAGE 80].
2. Pictures are natural to humans [PAGE 80].
3. Human brain assimilates pictorial information very fast [PAGE 80].
4. Glinert and Tanimoto report that 95 percent of the programmers in a comparison test preferred the iconic representation to the conventional textual representation of programs [HSIE 88].

3.2.2 Complexity Analysis

Since program understandability and program complexity are parallel concepts, researchers have attempted to measure program complexity. VIGRAM computes some of these complexity metrics, but we make no claims as to the validity of these metrics.

3.3 VIGRAM's Three Main Functions

VIGRAM has three main functions. The first two functions are for program understanding, the latter is for complexity analysis.

- 1) From a Pascal textual source program generate a visual "Plum Diagram" representing that source program. The plum diagram is more compact and layered for ease of comprehension.
- 2) Slice the plum diagram generated from 1) by suppressing all but a certain control structure(s), variable(s) and/or some input/output statements. This permits greater understanding of the program by showing selected visual traces through the program.
- 3) Perform complexity analysis of the Pascal source program : Berry-Meekings, Halstead, Weight of Difficulty, McCabe complexity metrics as well as a variety of counts i.e. the number of comment words, number of comment lines in initial block, number of tabs per line, percentage of

indentation tabs, number of simple type variables, and number of structure type variables.

3.4 Program Understanding

The more difficult a program is, the more difficult it is to maintain, hence understanding a program is a very important task of software maintenance. Graphical techniques are effective ways to understand a program. The reason is that the part of the human brain that assimilates pictorial information is much faster, and much older in evolutionary terms, than the part that interprets verbal information. Therefore, a graphic tool is more "readable" than a verbal one [PAGE 80]. VIGRAM augments the Pascal textual source program with a pictorial form called a "Plum Diagram". This provides a graphical view of a program, in addition to the textual one.

Some users may prefer the textual form of source code due to its familiarity. Because of this, VIGRAM allows a textual source program to be visible along with its pictorial form.

Moreover, a user can emphasize selected parts of code that contain certain object(s) and suppress the less interesting parts. These parts of code are called "slices". VIGRAM allows slicing on the control structure(s), variable(s), and/or some input/output statements.

VIGRAM makes the slices distinguishable by dimming the unsliced part. The idea is to isolate portions of program according to their behavior. This abstracting method allows the observation of

the specified slicing objects to be much faster, easier and clearer to understand.

Figure 3.1 shows hierarchical slicing of the Pascal procedure in Figure 2.4a on variables : "count" and "average". Note that "while" and "if-else" plums contain variable "count", therefore, they are also sliced.

Slicing can also be viewed as an "index" to rapidly find interesting object(s).

Slicing is a very powerful feature which also uses the advantage of pictures over text. Program is easier to understand and maintain when broken into smaller pieces [WEIS 81].

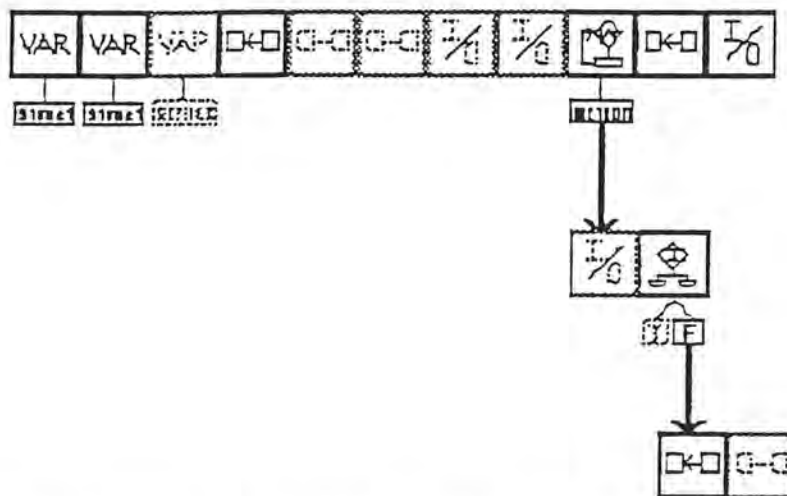


Figure 3.1 Hierarchical slicing of the Pascal procedure in Figure 2.4a on variables : "count" and "average".

3.5 Complexity Metric Analysis

VIGRAM computes the software complexity metrics of Berry-Meekings [HARR 86], Halstead [KEAR 86], weight of difficulty [BERN 84], McCabe [REDI 86], and a variety of counts i.e. the number of comment words, number of comment lines in initial block, number of tabs per line, percentage of indentation tabs, number of simple type variables and number of structure type variables.

The following describes the complexity metrics and counts that VIGRAM computes.

3.5.1 Weight of Difficulty

Berns [BERN 84] presented a technique to measure program difficulty, where he defines program difficulty as the sum of difficulties of its constituent elements. These elements can be quantified by the use of carefully selected weights and factors. He implemented this idea by assigning weights to the program elements of the FORTRAN language i.e. symbolic names, constant elements, operator elements, statement-type elements, etc. He reports results for FORTRAN programs, "...good scores are now considered to be less than 1,200". But people do not agree on the relative difficulties of understanding programs, i.e., there is no standard. "In the absence of a standard, we traditionally rely pragmatically on what we and our co-workers believe to be correct based upon our knowledge, experience, and intuition" [BERN 84].

Therefore, VIGRAM permits each user to assign weights to each construct found in Pascal. The default weights are all equal to one. The significance of the resulting score is not known.

3.5.2 Halstead complexity metrics

The following are Halstead measurement formulas [KEAR 86] :

$$\text{Volume (V)} = (N_1 + N_2) \text{Log}_2 (n_1+n_2)$$

$$\text{Program Difficulty (D)} = \frac{n_1 \times N_2}{2n_2}$$

$$\text{Effort (E)} = D \times V$$

where n_1 = number of unique operators

n_2 = number of unique operands

N_1 = total number of operators

N_2 = total number of operands

The validity of Halstead's metrics have been disputed by many researchers, hence we make no claims as to the significance of the volume, difficulty and effort metrics.

3.5.3 Berry-Meekings program characteristics

Percentage of comment lines, percentage of indentation spaces to all characters, percentage of blank lines, average number of nonblank characters per line, average number of spaces per line, number of symbolic constants used, symbolic constants and number of reserved words used are interesting metrics because they summarize the documentation quality of a program. Again, we make no claims about these metrics.

3.5.4 Other counts

The other counts are number of comment words, number of comment lines in initial block, number of tabs per line, percentage of indentation tabs, number of simple type variables, number of structure type variables and McCabe's total 'IF' count. See [REDI 86]. These counts, like other counts, are subjective and we make no claims to their validity. However, it is interesting to examine these metrics.

4. Using VIGRAM

4.1 Menu Reference

Figure 4.1 shows the overall menu items for VIGRAM.


 File Edit Code Source File Analyze			
About Plum Diagram...		Open...	Calculate Quality Metrics
About Visual Programming...		Close	Generate Full Chart
About Quality Metrics...			Slice Chart
Alarm Clock			Heading Weights
Chooser			Statement Weights
Control Panel			Constant Weights
.			Variable Weights
.			Operator Weights
.			

Figure 4.1 Overall menu items for VIGRAM.

Apple Menu

About Visual Programming...

Display the author name and VIGRAM's version.

About Quality Metrics...

Display a brief description of each complexity report produced from "Calculate Quality Metrics" item under "Analyze" Menu.

Source File Menu

Open...

Allow user to open an existing Pascal text file and select its procedures or functions to :

- Calculate Quality Metrics
- Draw a Full Chart
- Draw a Sliced Chart

Close

Close the current file.

Analyze Menu

Calculate Quality Metrics

Display : Berry-Meekings, Halstead, Weight of Difficulty, McCabe complexity metrics, number of comment words, number of comment lines in initial block, number of tabs per line, percentage of indentation tabs, number of simple type variables, and number of structure type variables for a Pascal text file procedure.

Generate Full Chart

Draw a Plum Diagram of a Pascal text file procedure.

Slice Chart

Slice the current Plum Diagram on certain control structure(s), variable(s) and/or some input/output statements.

Heading Weights

Allow user to assign weights to program heading, definition and declaration constructs : each program (or procedure or function) keyword, each parameter, definition and declaration found.

Statement Weights

Allow user to assign weights to each simple statement, procedure call, begin, case, else, end, for, if, repeat, while, and with keywords found.

Constant Weights

Allow user to assign weights to each local integer, real, string, unit global, nil, and true & false constants found.

Variable Weights

Allow user to assign weights to each local boolean, integer, longint, real, char, structure & others, unit global variable and parameter, and each function call, non-declared identifier, and with clause found.

Operator Weights

Allow user to assign weights to each of these operators found :

+, -, *, /, DIV, MOD, =, <, <=, >, >=, IN, OR, AND, NOT, @.

4.2 Tutorial

To use VIGRAM we first open a Pascal text file containing either a standard Pascal source code program or a unit of Lightspeed Pascal for the Macintosh. To open a Pascal source file pull down "Open..." item under "Source File" menu.

After opening a source file, VIGRAM will automatically show a source file and the list of procedure and function names found in that source file in separate windows. These windows can be resized, moved and scrolled. Figure 4.2 shows these two windows after the source file named "average" is opened.

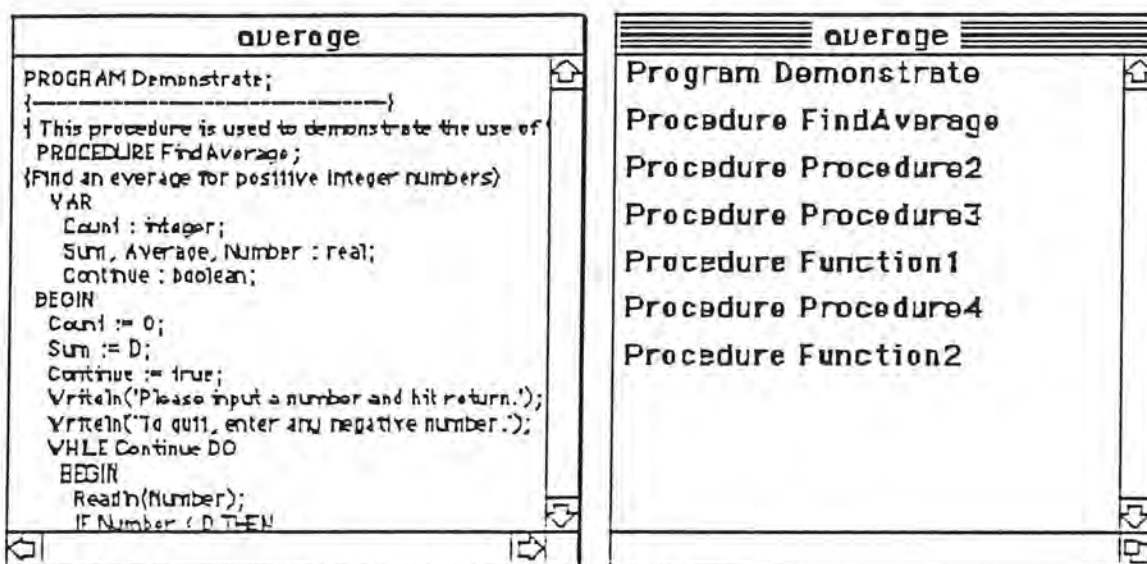


Figure 4.2 Source file window and procedure & function list window.

Select a procedure or function to be analyzed by mouse-clicking on that procedure or function name listed in the right window. The procedure or function name chosen will be highlighted to show the selection. Another window will pop up to show the selected procedure textual source code. See Figure 4.3. This window can also be resized, moved, scrolled and closed.

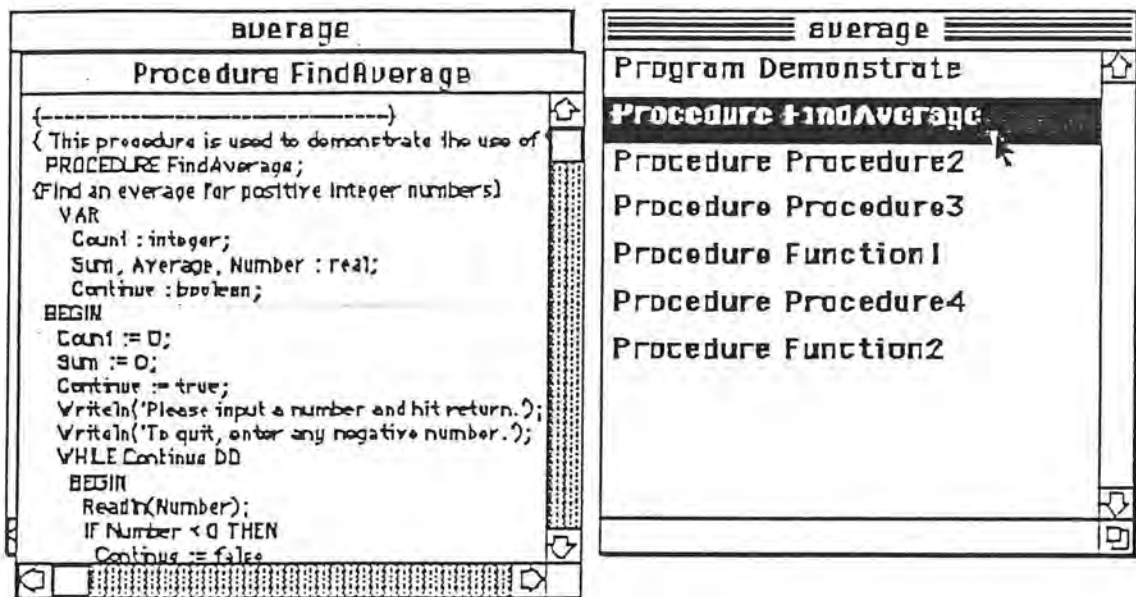


Figure 4.3 Selecting a procedure or function to analyze.

Now the user can perform 2 operations : 1) generate the plum diagram and 2) perform complexity analysis. First we generate the plum diagram then do slicing.

1) Generate Plum Diagram

To generate the visual "Plum Diagram" representing the source program, pull down "Generate Full Chart" item under "Analyze" menu. Figure 4.4 shows the plum diagram generated from the procedure in Figure 2.4a. For more details about manipulating plum diagrams see [HSIE 88].

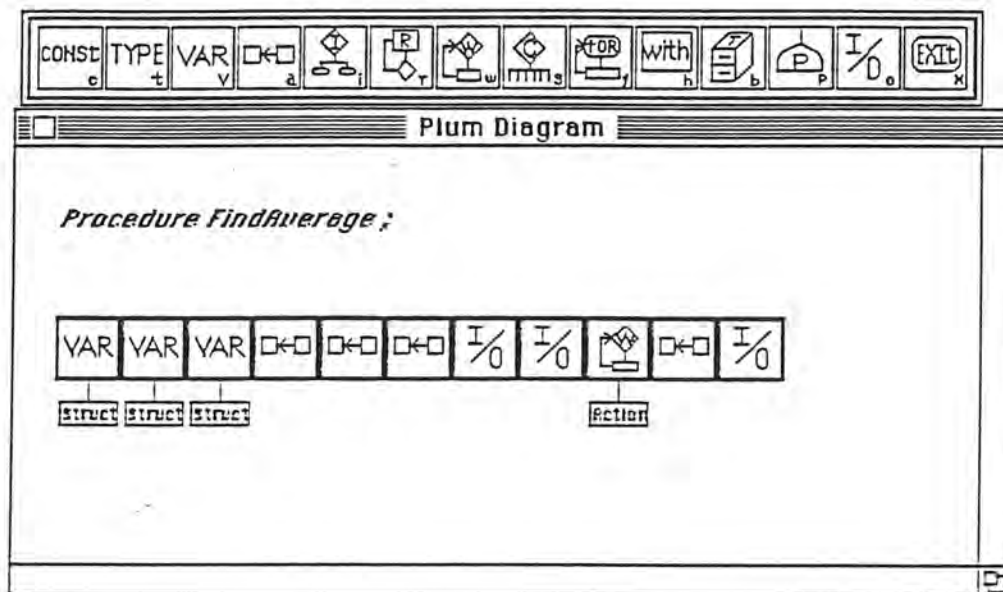


Figure 4.4 Plum Diagram for procedure "FindAverage" in Figure 2.4a.

2) Perform Complexity Analysis

For the "weight of difficulty" metric, the user can assign weights corresponding to each Pascal element of the procedure : procedure heading-definition-declaration, constants, variables, statements and operators by selecting "Heading Weights", "Statement Weights", "Constant Weights", "Variable Weights", and "Operator Weights" items from the "Analysis" menu. The default weights for all elements is one. Once set, VIGRAM remembers the weights until they are reset or until the user quits the application. Dialogs for weight setting are shown in Figures 4.5 through 4.9.

To perform complexity analysis, select item "Calculate Quality Metrics" from menu "Analyze". Figure 4.10 shows a sample report from the analysis. Figure 4.11 and 4.12 shows brief descriptions of a complexity metric report under "About Quality Metrics..." of the "apple" menu.

Heading, Definition & Declaration	
PROGRAM	1.00
PROCEDURE	1.00
FUNCTION	1.00
Parameter	1.00
Constant Definition	1.00
Type Definition	1.00
Variable Declaration	1.00
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

Figure 4.5 Heading, Definition & Declaration Weight Setting Dialog.

Statements		
Control Statements		
	BEGIN	1.00
	CASE...OF	1.00
	ELSE	1.00
	END	1.00
	FOR...TO (DOWNTO)	1.00
	IF...THEN	1.00
	REPEAT...UNTIL	1.00
	WHILE...DO	1.00
	WITH...DO	1.00
Simple Statements	1.00	
Procedure Calls	1.00	
<input type="button" value="OK"/> <input type="button" value="Cancel"/>		

Figure 4.6 Statement Weight Setting Dialog.

Constants	
Local Constants	
Integer	1.00
Real	1.00
String	1.00
Unit Global Constants	1.00
NIL	1.00
TRUE & FALSE	1.00
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

Figure 4.7 Constant Weight Setting Dialog.

Variables & Parameters	
Local Variables and Parameters	
Boolean	1.00
Integer	1.00
Longint	1.00
Real	1.00
Char	1.00
Structured & Others	1.00
Unit Global Variables	1.00
Function Calls & Non-declared Identifiers & 'WITH' clauses	1.00
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

Figure 4.8 Variable Weight Setting Dialog.

Operators			
Arithmetic and Set Operators		Relational and Set Operators	
'+' Addition, Identity	<input type="text" value="1.00"/>	=, <>, <, >, <=, >=, IN	<input type="text" value="1.00"/>
'-' Subtraction, Negation	<input type="text" value="1.00"/>	Boolean Operator	
'*' Multiplication	<input type="text" value="1.00"/>	OR	<input type="text" value="1.00"/>
'/' Division	<input type="text" value="1.00"/>	AND	<input type="text" value="1.00"/>
DIV	<input type="text" value="1.00"/>	NOT	<input type="text" value="1.00"/>
MOD	<input type="text" value="1.00"/>	@ Operator	<input type="text" value="1.00"/>
<input type="button" value="OK"/>		<input type="button" value="Cancel"/>	

Figure 4.9 Operator Weight Setting Dialog.

Report for Procedure FindAverage			
Weight of Difficulty	46.00	Berry & Meekings	
Comment		% of Comment Lines	10.71
Total Words	18	% of Indentation Spaces	0
Lines in Initial Block	3	% of Blank Lines	0
Tab		Characters per Line	18.28
Tabs per Line	2.68	Spaces per Line	1.86
% of Indentation Tabs	11.74	Symbolic Constants	0
Variable		Reserved Words	18
Simple Type	16	Halstead	
Structured Type	0	Volume	163.50
McCabe		Difficulty	8.5
Total 'IF'	1	Effort	1316.16
		<input type="button" value="OK"/>	

Figure 4.10 Sample complexity metric report.

About Quality Metrics

Weight of Difficulty	= Summation of all Weights
Comment :	
Total Words	= total comment words found
Lines in Initial Block	= total comment lines before or after procedure or function heading
Tabs :	
Tabs per Line	= $\frac{\text{total tabs}}{\text{total lines}}$
Percent of Indentation Tabs	= $\frac{\text{indentation tabs}}{\text{file length}} \times 100\%$
Variables :	
Simple Type	= total local simple type variables found (simple types are char, real, integer, longint and boolean)
Structured Type	= total local structured type variables found (all local variables except simple type variables)
McCabe's Total 'IF'	= total 'IF' keyword found

NEXT

QUIT

Figure 4.11 Description of a complexity metric report under "About Quality Metrics..." item of "apple" menu.

About Quality Metrics

Berry & Meekings :

Percent of Comment Lines	=	$\frac{\text{total comment lines}}{\text{total lines}}$	x 100%
Percent of Indentation Spaces	=	$\frac{\text{indentation spaces}}{\text{file length}}$	x 100%
Percent of Blank Lines	=	$\frac{\text{total blank lines}}{\text{total lines}}$	x 100%
Characters per Line	=	$\frac{\text{file length} - \text{total spaces} - \text{total tabs}}{\text{total lines}}$	
Spaces per Line	=	$\frac{\text{total spaces}}{\text{total lines}}$	x 100%
Symbolic Constants	=	total number of local constants found	
Reserved Words	=	total number of Pascal reserved words found	

Halstead :

Volume (V)	=	$(N_1 + N_2) \text{Log}_2 (n_1+n_2)$	
Program Difficulty (D)	=	$\frac{n_1 \times N_2}{2n_2}$	
Effort (E)	=	$D \times V$	

where n_1		= number of unique operators
n_2		= number of unique operands
N_1		= total number of operators
N_2		= total number of operands

PREVIOUS

QUIT

Figure 4.12 Description of a complexity metric report under "About Quality Metrics..." item of "apple" menu.

3) Slice Plum Diagram

To slice the plum diagram, select the "Slice Chart" item under "Analyze" menu. The slice-criterion dialog will pop up, see Figure 4.13. Check each control or I/O structure to be emphasized. One or more variable can also be selected from the list on the right. Only plums containing or leading to these items will be highlighted in the plum diagram. Figure 4.14 shows the plum diagram in Figure 4.12 obtained from slicing the program in Figure 2.4a using the criterion in Figure 4.13. Slicing and unslicing can be performed at any level of the original plum diagram. Slice and unslice are toggle switches.

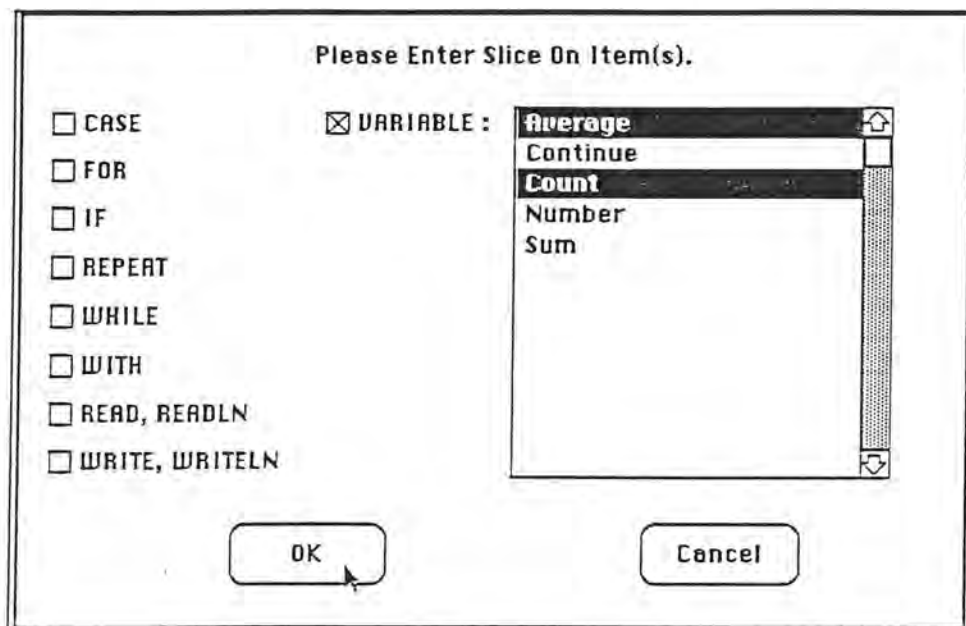


Figure 4.13 Slice-criterion dialog.

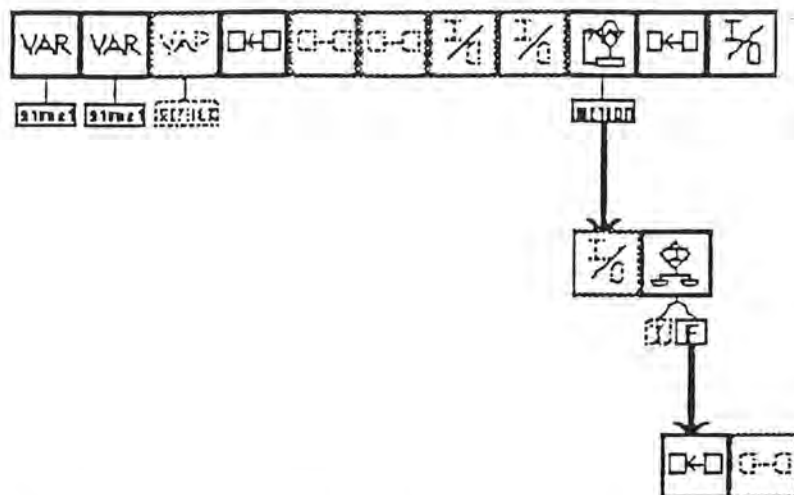


Figure 4.14 A hierarchical sliced plum diagram obtained from settings shown in Figure 4.13.

To close a Pascal source file pull down "Close" item under "Source File" menu. Once a source file is closed, we can open another file.

5 Implementation

The source code is stored in RAM all at once and handled by "TextEdit", a Macintosh toolbox routine. The maximum text length is 32K bytes [APPL 86], thus, programs must be less than 32K bytes in length.

5.1 Data Structure

The main data structure of VIGRAM is an enumerated type for each token found when doing a lexical analysis. The data structure for a plum diagram is the same as the one of P.D. Editor [HSIE 88], except that we have added a display flag for slicing. See Figure 5.1.

```
token_type = (
    {Keywords}
    ARRAYkw, BEGINkw, CONSTkw, DOkw, DOWNTOKw, ELSEkw,
    ENDkw, ENDDOTkw, EXTERNALkw, FILEkw, FORWARDkw,
    FUNCTIONkw, GOTOkw, INLINEkw, INTERFACEkw,
    IMPLEMENTATIONkw, LABELkw, NILkw, OFkw,
    OTHERWISEkw, PACKEDkw, PROCEDUREkw, PROGRAMkw,
    RECORDkw, SETkw, THENkw, TOKw, TYPEkw, UNITkw,
    UNTILkw, USESkw, VARkw, BOOLEANkw, CHARkw,
    INTEGERkw, LONGINTkw, REALkw, STRINGkw, TEXTkw,

    {String constant i.e. "stringA"}
    STRINGconst,

    {Keywords}
    TRUE_FALSEkw, IFkw, FORkw, WHILEkw, WITHkw,
    REPEATkw, CASEkw,

    {Operators}
    PLUSop, MINUSop, Orop, TIMEop, SLASHop, DIVop, MODop,
    ANDop, NOTop, LTop, LEop, EQop, NEop, GTop, GEop, INop,
    ADDop, MULop, REop, ADDRESSop,

    {Special character(s)}
    ARROW, ASSIGN, COMMA, DOTDOT, LB, LP, RP, RB, SEMI,
```



```

COLON, PERIOD,

{Identifier}
IDENT,

{integer, real}
INT, NUM,

{ Comments : " (*, {, *), } " }
OpenStarCom, OpenBraceCom, CloseStarCom, CloseBraceCom,

{white characters}
SPACE, NEWLINE, TAB,

{Heximal digit error or other error}
HEXERR,

{For Heading weights : Parameter, Constant Definition,
Type Definition, Variable Declaration}
Para, CDef, TDef, VDef,

{For Statement Weights : Simple Statement, Procedure Call}
SimpSt, PCall,

{For Constants Weights : Integer Const, Real Const, String
Const, Unit Global Constant}
IntCnst, RealCnst, StrCnst, UnitGloCnst,

{For Variable Weights : Boolean Variable, Integer Variable,
Longint Variable, Real Variable, Char Variable, Structured
& others variable, Unit Global Variable, Function Calls &
Non-declared Identifiers & 'WITH'clauses }
BolVar, IntVar, LintVar, RealVar, ChVar, StructOthVar,
UnitGloVar, NonDecID_FCallVar
);

```

Figure 5.1 The main data structure : token-type.

5.2 Possible Extensions.

- 1) Other graphical forms can be tried to replace the Plum Diagram.
- 2) Rewrite the lexical analysis part in Assembly language to reduce the runtime.
- 3) Experiment on the weight of difficulty to find the "best" default weights.

5.3 Requirements and limitations.

- 1) VIGRAM accepts a Pascal source code unit or program. VIGRAM recognizes standard Pascal programs and Lightspeed Pascal units.
- 2) The maximum text length is 32K bytes.
- 3) The source code must be syntactically correct. This is for runtime time-reduction. An alert is shown when a syntax error occurs, and VIGRAM halts.
- 4) VIGRAM doesn't support nested procedures or functions.

5.4 Application Statistics

1. Number of lines of source code	10,277
2. VIGRAM application size	145 K
3. Resource size	
3.1 Uncompiled (.R)	80.5 K
3.2 Compiled (.Rsrc)	25.5 K
4. Number of Units with P.D. Editor	16

6. Summary

6.1 Program Understanding

We have presented a program understanding and complexity metric analysis tool. VIGRAM's main aim, to make programs easier to understand, has not been tested, but we believe it is promising for the following reasons :

- We use a graphical form to augment, rather than replace the textual form. The results should be no worse than text only.
- Slicing can be viewed as "indexing" to rapidly find interesting object(s).

After using VIGRAM as an understanding tool, we can modify a program using P.D. Editor.

6.2 Complexity Metrics

Figure 6.1 shows the table we got from running VIGRAM's three units, on nineteen procedures (functions) to compare two program difficulties we compute : "Weight of Difficulty" and "Halstead's Difficulty". On Figure 6.1, column :

- 1 : The rank of "Weight of Difficulty" in column 4.
- 2 : The rank of "Halstead's Difficulty" in column 5.
- 3 : The difference of column 1 from column 2.

- 4 : Weight of Difficulty applied from [BERN84] with the default weight one.
- 5 : Halstead's Difficulty = $\frac{n_1 \times N_2}{2n_2}$ where N_2 , n_1 and n_2 are of columns 9 to 11.
- 6 : Total words without comment words. By words, we mean Syntactic tokens, e.g. $N_1 + N_2 +$ "reserved words" + "declaration variables" + "closing parenthesis" + "opening bracket" + "closing bracket" + "dot dot token" + colon + string constant + semicolon.
- 7 : Total Reserved words.
- 8-11 : N_1 = total number of operators
- N_2 = total number of operands
- n_1 = number of unique operators
- n_2 = number of unique operands

Figure 6.2 shows that the number of total words is a factor of "Weight of Difficulty".

1	2	Rank Comparison	Weight of Dif.	Hal's Dif.	Words	Reserved words	N1	N2	n1	n2
1	1	0	7.00	0.00	15	3	0	0	0	0
2	3	1	22.00	2.75	54	4	11	11	2	4
3	4	1	25.00	3.14	54	15	7	11	4	7
4	14	10	37.00	10.00	79	15	20	20	9	9
5	5	0	39.00	3.50	74	14	15	21	4	12
6	8	2	40.00	6.00	86	12	21	24	6	12
7	9	2	43.00	7.27	94	28	20	20	8	12
8	7	-1	48.00	5.41	93	23	24	23	8	17
9	2	-7	55.00	2.10	90	18	21	35	3	25
10	10	0	62.00	8.06	135	27	37	31	13	25
11	6	-5	78.00	5.03	175	12	50	52	6	31
12	11	-1	86.00	8.50	166	28	42	51	8	24
13	15	2	90.00	11.80	190	26	57	59	10	25
14	13	-1	110.00	8.88	206	26	55	63	11	39
15	16	-1	144.00	16.63	304	35	101	110	13	43
16	12	-4	152.00	8.62	318	51	87	96	7	39
17	18	1	202.00	30.00	439	76	151	110	18	33
18	17	-1	391.00	22.47	791	121	214	230	17	87
19	19	0	795.00	33.01	1299	297	403	471	15	107

Figure 6.1 Difficulty Comparison Table.

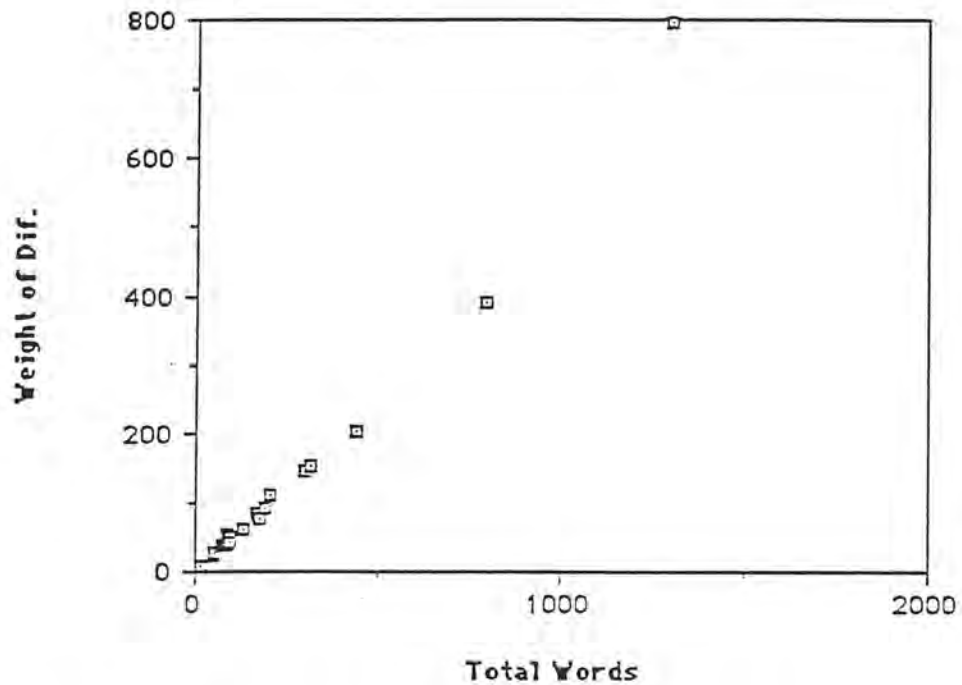


Figure 6.2 Total words is a factor of "Weight of Difficulty".

From the table in Figure 6.1, there are four out of nineteen significant rank differences : (10, -7, -5, -4 in column 3). Four procedures have the same rank (four zeros in column 3); fifteen procedures differ by a small amount : (-1, 0, 1, 2 in column 3). Weight of difficulty is in the opposite direction from Halstead's Difficulty metric for procedures ranked 7 and 8 according to Weight of Difficulty.

Figure 6.3 is a scatter graph shows the relation between "Weight of Difficulty" and "Halstead's Difficulty". There is a strong relation between these two difficulties, even though some conflicts are still found.

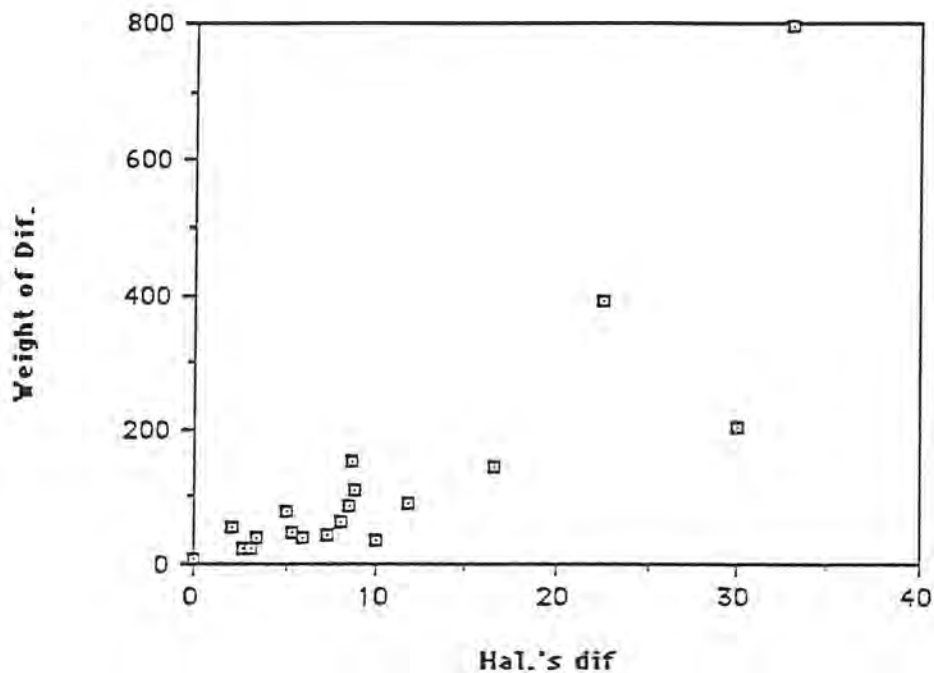


Figure 6.3 Scatter Graph showing relation between "Weight of Difficulty" and "Halstead's Difficulty".

7. Bibliography

[APPL86] Apple Computer, Inc.

"Inside Macintosh", May 1986, Volume 1, pp.1378.

[BERN84] G. M. Berns,

"Assessing Software Maintainability", *Communications of The ACM*, January 1984, Volume 27 Number 1, pp. 14-23.

[HARR86] W. Harrison and C.R. Cook,

"A Note on The Berry-Meekings Style Metric",
Communications of The ACM, February 1986 Volume 29
Number 2, pp.123-125.

[HSIE88] C. Hsieh,

"A Graphical Editor for Pascal Programming on
Macintosh", Research Paper, Department of Computer
Science, Oregon State University, March 1988.

[KEAR86] J. K. Kearney, R. L. Sedlmeyer, W. B. Thompsom,
M. A. Gray, and M. A. Adler;

"Software Complexity Measurement", *Communications of
The ACM*, November 1986 Volume 29 Number 11,
pp.1044-1050.

[LEWI87] T. G. Lewis,

"CASE Computer-Aided Software Engineering", Computer
Science Department Oregon State University, Corvallis,
O.R. 97331.

- [PAGE80] M. Page-Jones,
"The Practical Guide to Structured Systems Design",
Yourdon Press, 1501 Broadway, New York, NY 10036.
- [REDI86] K. A. Redish and W. F. Smyth,
"Program Style Analysis: A Natural By-product of
Program Compilation", *Communications of The ACM*,
February 1986 Volume 29 Number 2, pp.126-133.
- [WEIS81] M. Weiser,
"Program Slicing", *Proceedings of the Fifth International
Conference on Software Engineering*, 1981, pp. 439-449.
- [YANG88] S. Yang, T.G. Lewis, and C. Hsieh,
"OSU: Integrating CASE and UIMS", Computer Science
Department, Oregon State University, Corvallis, OR 97331,
1988.