THE ANALYSIS OF D_i, A DETAILED DESIGN METRIC,
ON LARGE SCALE SOFTWARE

A THESIS
SUBMITTED TO THE GRADUATE SCHOOL
OF BALL STATE UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
OF MASTER OF SCIENCES
BY
PATRICK DREW MCDANIEL

ADVISOR - WAYNE M. ZAGE

Committee Approval:

bWayne M Zage — 4/30/91

Committee Chairman — Date

Clint P. Swilling — 4-30-91

Committee Member — Date

Dolores M. Zage — 4-30-91

Committee Member — Date

Department Head Approval:

Roy L. McCormick — 4-30-91

Head of Department — Date

Graduate Office Approval:

Dean of Graduate School — Date

THE ANALYSIS OF D_i, A DETAILED DESIGN METRIC, ON LARGE-SCALE SOFTWARE

A THESIS

SUBMITTED TO THE GRADUATE SCHOOL

OF BALL STATE UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCES

BY

PATRICK DREW MCDANIEL

ADVISOR - WAYNE M. ZAGE

BALL STATE UNIVERSITY

MUNCIE, INDIANA

MAY 1991

Acknowledgements

I wish to thank first and foremost the members of my thesis committee, Dr. Wayne Zage, Chairman, Dr. Clinton Fuelling, and Profesor Dolores Zage.

I also would like to thank Mary Meadows for her help in acquiring the resources needed to complete this thesis as well in keeping my perspective, the members of the Ball State Design Metrics team for their suggestions, and John Cassidy for his advice and help.

I would like to thank my family for their support throughout my education.

I would like give a special thanks to my father, without whose help this thesis would not have been possible.

Table of Contents

List of Tables

List of Figures

Introduction

Counter to the beliefs of computer users today, most software being developed today contains many errors. The reasons for this predicament is in the failure of software developers to use standard practices that one would find in other industries. Quality assurance is just one area where these developers fall short. The software engineering sciences are aimed at giving the same level of productivity and quality that one might find in other technologies. Design metrics are intended to be used by the project managers to make sure that quality of product is maintained, whereas models of software development are used to plan and track a project.

As any new technology is used, methodologies, standards, and efficiency measures are developed. The software engineering sciences are intended to develop these standards for software development and maintenance. Specifically, metrics are intended to be used as professional measurements for software. Technologies that do not allow the comparison of dissimilar objects by common attributes cannot fully be understood, resulting in suspect methodologies and inconsistent

standards. Software engineering is no acceptation. In software engineering projects can be easily compared by size or functionality, but nothing meaningful can be said about the relative quality of the two projects without further investigation. Design metrics were researched with the intention of finding a yardstick for software quality.

The central problem cited by people that do not believe in the quality metrics, is not the need but validity of the metrics. In the article "Complete Solution to Measurement Problem", Michael Evangelist argues that the metrics being developed today are bankrupt. He feels that the researchers are not interested in theoretical bases of these metrics, but of only the results. It the intention of this thesis to give the theoretical bases for the D_i metric as well as give empirical evidence for the support of the D_i metric as a quality measure. Projects taken from the software engineering class at ball state university were analyzed. These projects provide strong evidence that these measures actually indicate software quality by highlighting error-prone modules. The success of the metric warrants further study, specifically on large-scale software.

Having obtained a sample portion of a large-scale system, the research team sought to prove or disprove D_i's viability as a quality metric. A

system for collecting metrics on existing ADA code was developed. This system was verified and used collect the metric counts, which were in turn compared to the error reports supplied by the Computer Science Corporation, the author company of the system.

Due to circumstances beyond the control of the researchers, the amount of information about the project being analyzed was minimal. This problem led to irreconcilable elements within the system. This not withstanding, the data collected supported the use of the D_i metric, however, due to the extenuating circumstances, I feel that this study should be replicated on a separate set of data.

In the end analysis, the D_i metric was found to be good indicator of software error, and these results warrants further study. In the pages that follow, the background, analysis techniques, and results will be outlined.

Chapter 1

D_i Definition and Background

Why Design Metrics?

Large-scale software has been developed since the introduction of high powered computers into the workplace. As these systems grow larger, the ability of the designer to manage and understand the amount of information encompassed within will decrease. In response to this problem computer scientists have begun research in a field known as software engineering. In this new field, standard ways of analyzing requirements, testing, designing, and other tasks of project development have been studied.

In addressing the inherent problems with software development, statistical models and software metrics have been researched. With these models the developers can obtain information about a developing system and make managerial or technical decisions about the project. The design metrics that are being developed at Ball State University are intended to estimate the quality of a given design. This quality can be determined at the design phase, giving the developers an early indication of project status. The metrics also can help in the making of decisions about the design, limit complexity, and identify potential problem areas in the design.

D(G) Definition

Software metrics can be used to quantify some characteristic of the software development process or product. The Ball State Design Metrics Research Team has been investigating a design quality metric. This metric will determine the quality of a system design by taking measurements from architectural and detailed design artifacts. This metric will highlight "stress point" modules, where it is determined that pockets of errors may occur. After collecting these counts, the developers can redesign or make allowances for these problem areas.

The D(G) metric contains one external and one internal metric component. The external metric, D_e, explained in detail later, measures the stability of an architectural design. The internal metric, D_i, measures the quality of the detailed, design of a module. The two metrics, D_e and D_i, are used in a linear equation to calculate D(G), a measurement of total design quality.

For a structured design G, the design quality metric D(G) has the form

$$D(G) = k_1(D_e) + k_2(D_i).$$

In this equation k_1 and k_2 are constants and D_e and D_i are the external and the internal design quality metric counts for the design G. All module counts are collected similarly, and the average and standard deviation of the counts are calculated. Any module whose count is one

standard deviation above the mean is determined to be an "outlier". It is these outliers that we suggest may contain errors.

It is not necessary for the designers to wait until the entire design process is over. Outliers can be determined at the end the architectural and detailed design phases by identifying outliers with the D_e or D_i counts alone. This process of design-calculate-redesign can continue until the project team is satisfied with the system design.

Figure 1-1 shows how this iterative process is executed within the software design waterfall model.

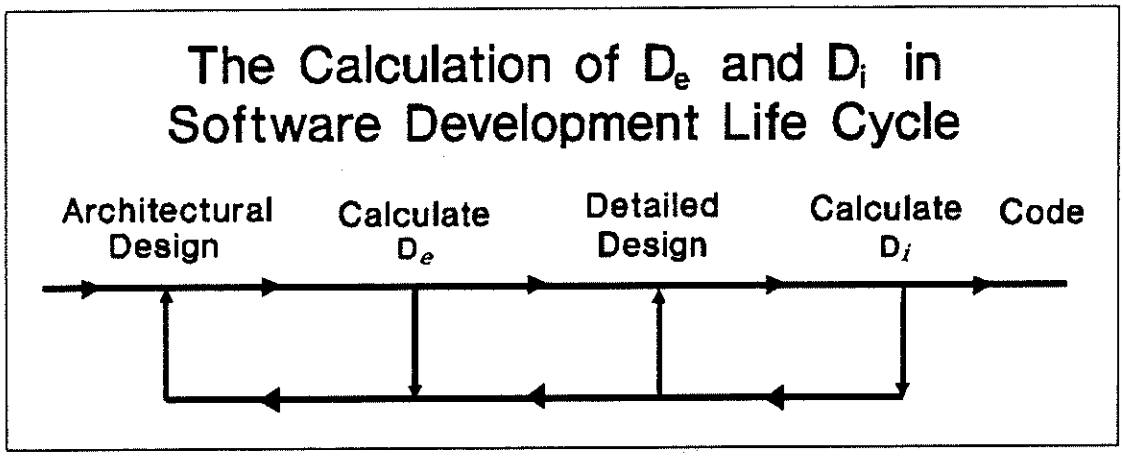

Figure 1-1

D_e History

Dr. Wayne Zage and Professor Dolores Zage began research in the Fall of 1989 on the D_e metric by analyzing standard design notations. This analysis led to the use of a graph theoretic approach to analyze the

attributes of software designs. The metric measures these attributes of design notations relating these metrics to the design principles of complexity, coupling, modularity, and size. The thesis of the research is that the combination of these attribute measurements will determine the quality of design by predicting error-prone modules.

D_i History

In January of 1990 the design metrics research team began researching the detailed design metric D_i. After reviewing the D(G) metric and its relation to D_i, members of the team reviewed books and periodicals on detailed design and code metrics. From these resources, the team selected McCabe's cyclometric complexity measure (V(G)) and source lines of code (SLOC) to evaluate. After exchanging information and observations, I selected the preliminary D_i for testing. A set of projects from the software engineering class at Ball State University was selected for the analysis of the metrics. During this pilot study the team calibrated the weights of the components, which resulted in the metrics used on a large-scale software project from Computer Science Corporation called STANFINS. These tests resulted in the pilot study listed in chapter 3.

D_o Definition

The calculation of D_e is the sum of two components. The first component measures the amount of data flowing through the modules, whereas the second measures the number of paths through the module. More precisely,

$$D_e = (\text{weighted-inflows} * \text{weighted-outflows}) + (\text{fan-in} * \text{fan-out})$$

where:

Weighted Inflows

are the actual number of data items flowing into the module. This is the number of parameters passed to the module plus the number of global data items being accessed by the module.

Weighted Outflows

are the number of data items that the module passes to other modules. Note that this component is tied closely to the central calls component of the D_i metric.

Fan In

is the number of superordinate modules of the module. This is equal to the number of modules that make calls to this module within the system.

Fan Out

This is the number of subordinate modules of the module. This is equal to the number of unique modules called by the module.

The results of the pilot study, outlined in chapter 3, support the claim that this metric is a good predictor of architectural design quality by highlighting the error-prone modules in a system.

D_i Definition

During the pilot study performed on the CS680 projects, a preliminary D_i metric was selected. After reviewing other related research in the field, we felt that the traditional metrics could not capture the full complexity of the detailed design.

It was the observation of the team that a metric that measured the possible locations for errors in a software project might adequately capture detailed design quality. Informal interviews were conducted with various members of the Integrated Technologies Research and Development Department. When asked where software errors occur, these developers responded with three distinct categories. These categories include, input and output sections, complex data structure manipulation sections, and calls to other modules. It was then deemed reasonable that the measurement of these occurrences might successfully capture the complexity of the modules. This line of reasoning led us to the following preliminary D_i definition.

$$D_i = w_1(CC) + w_2(DSM) + w_3(I/O)$$

Where:

Central Calls (CC)

are procedure or function invocations. These are calls to non-library modules.

Data Structure Manipulations (DSM)

are references to complex data types. In this definition, a complex data type is one that uses indirect addressing. (e.g. pointer, array, record, typedef, ...)

Input/Output (I/O)

are external device accesses. This can be any read/write operation to a device (e.g. read file, stdin, stdout, port, device, ...)

and $w_1 = w_2 = w_3 = 1$.

The unit weighting scheme was selected at first for simplicity, but the calibration of these factors will affect the metric counts dramatically.

D(G) Example

The following paragraphs outline the D(G) metric analysis process. Note that this example uses a selected segment of a system, and is meant to illustrate the collection of the design metrics on a given module. The post-collection statistical analysis is outlined later in this chapter.

After the architectural design phase of a system, the developers will have obtained the system specifications, a structure chart, data flows, functional descriptions of modules, interface descriptions, and the data

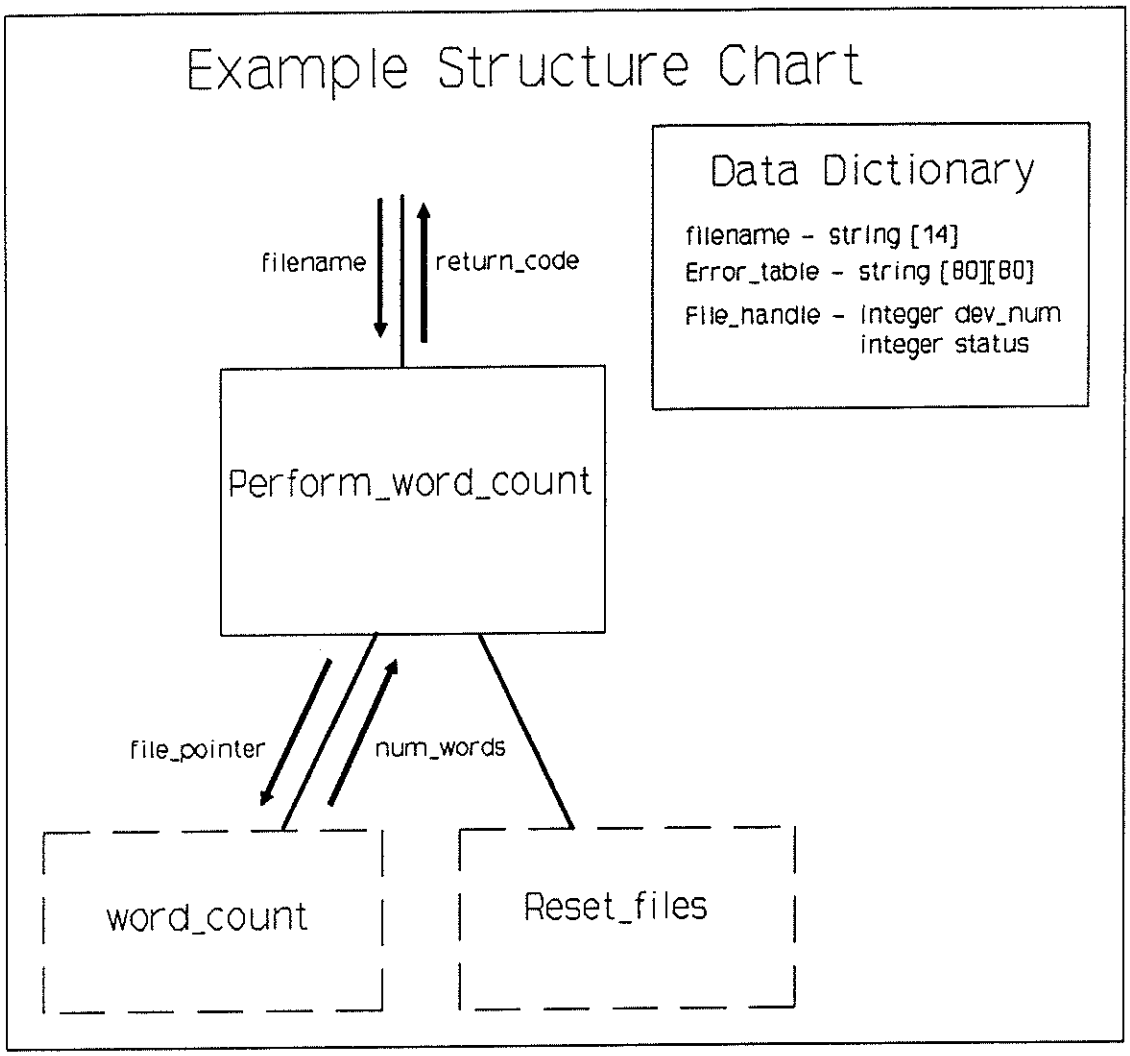

Figure 1-2

definitions. It is from these items that the external D_e metric is calculated. Figure 1-2 shows the structure chart for the module perform_word_count.

Recall:

D_e = (weighted-inflows * weighted-outflows) +

(fan-in * fan-out)

The weighted-inflows component of the D_e metric measures the amount of data passing into a module. The weighting of a inflow is determined by the number of atomic elements contained in a passed data item. A structure or record may contain many atomic elements, each of which is given a value of one. A determination of the number of atomic elements a data item contains may be obtained from the data dictionary. In our example, the inflowing data items **filename** and **num_words** are both atomic elements, so the value of weighted-inflows is 2.

The weighted-outflows count for a module is the sum of data-items exiting the module, weighted by the number of atomic data items contained in their definitions. A determination of the weight of a given weighted-outflow may be obtained in the same way as a weighted-inflow. The example module contains two outflows. The **return_code** outflow is a single integer and is weighted 1. The **file_pointer** outflow definition contains two integers and is weighted 2. Adding these two weights results in a outflow count of 3.

The third component, fan-in, is the sum of the superordinate modules of the module. In our example, there is only one superordinate module of the module, so the fan-in count is 1.

The fan-out component of the D_e metric measures the subordinate modules of the module. Two modules, **reset_files** and **word_count**, are subordinates to the module, thus giving us a fan-out count of 2.

With all of these counts collected, we can now calculate the D_e count for this module. We plug these counts into the equation,

D_e = (weighted-inflows * weighted-outflows) + (fan-in * fan-out)

D_e = (2 * 3) + (2 * 1)

D_e = 8

giving a D_e value of 8 for this module. Repeating this process for all modules in the system gives us a count population to analyze.

At the detailed design phase, the designers will have all resources obtained on or before architectural design plus algorithms selected, a complete data dictionary, and some detailed design documents. From these documents and the previous D_e counts we can calculate the D_i and D(G) metrics.

In Figure 1-3, a detailed design document for the D_e example module is presented. Recall that

$D_i = w_1(CC) + w_2(DSM) + w_3(I/O)$

where

$w_1 = w_2 = w_3 = 1.$

To calculate the D_i metric, we count the number of occurrences of each component. These counts can be collected manually or by an automated

Example Detailed Design

```
1    Gobal Error_table
2    Procedure integer Perform_word_count ( filename )
3    Begin
4       local file_handle file_pointer;
5       local integer Number_words
6       if (not        file_pointer = open ( filename ) )
7          Begin
8             Error_table [1] = "File Open Error "
9           End
10      else
11         Begin
12            if ( Number_words = word_count(file_pointer) = 0 )
13               Begin
14                  Error_table [2] = "File Read Error "
15                  Close ( File_pointer )
16               End
17            else
18               Begin
19                  Print ( " %d Number Words .", Number_words )
20                  Print ( " Successful Termination." )
21                  Close ( File_pointer )
22                  Reset_files ( void )
23                  Return ( Number_words )
24               End
25         End
26      Return ( 0 )
27   End
```

Figure 1-3

tool[1]. By locating the occurrences and applying the D_i equation to the resulting counts, one can obtain the count for the module. The central call (CC) count for a given module is acquired by counting the number

[1] As of February 1991, a design tool to collect the D_i metric has not been completed, but is the subject of research within the design metrics research team.

of calls to non-library modules. A non-library module is defined as a module that has not been designed specifically for the system under development. This determination is made, for the libraries have their own development and testing schedules, which may or may not be related to the current system. In our example there are two such central calls, one to **word_count** (located on line 12) and another to **reset_files** (located on line 21). This module thus has a central call count of 2.

A data structure manipulation is defined as any indirect access to a data item. An indirect access is any access to a data item that requires some addressing calculation. Examples of data structure manipulations are an access to array, structure, or record element, a pointer, or any inheritance class elements of a data item. The scope of a data item has no bearing on the ability of a data item being a data structure manipulation. The detailed design of our example contains two DSMs, both being accesses to the global **error_table** (lines 8 and 14). These accesses give this module a data structure manipulation count of 2.

The input/output count for a given module is the number of external device accesses within that module. A external device is defined as any resource which requires a device driver. Examples of external devices are printers, modems, keyboards, disks, monitors, and sound boards. Note that these accesses need not be reads or writes, they may

include utility interrupts or device status changes. The example includes two writes to standard out (**print** located on lines 18 and 19), and three utility calls, one to **open** (located on line 6) and two file closures (located on lines 15 and 20). This gives us a input/output count of 5 for this module.

Having collected these counts, we can now use them in the D_i equation as follows :

$$D_i = w_1(CC) + w_2(DSM) + w_3(I/O)$$

where

$$w_1 = w_2 = w_3 = 1$$

$$D_i = 1*(2) + 1*(2) + 1*(5)$$

$$D_i = 9$$

With the D_e and D_i counts we can calculate the D(G) metric. The resulting value will give the relative design quality of this module. The calculations are as follows :

$$D(G) = k_1(D_e) + k_2(D_i)$$

where

$$k_1 = k_2 = 1$$

$$D(G) = 1*(8) + 1*(9)$$

$$D(G) = 17$$

Note that this count has no significant meaning without the counts of the other modules in the system. The value is the design quality

relative to the other modules. The statistical analysis of this population will highlight the "stress points" within the system.

Statistical Analysis Example

Once the counts for the modules have been calculated, it is necessary to analyze the data to determine the outlier modules. Note that all the counts need not be collected to determine outlier modules. After the architectural design phase, the D_e counts are collected and outliers are highlighted. These highlighted modules can be redesigned and the counts recalculated. This process is repeated until the design team is satisfied with the architectural design.

In the example population, listed in Table 1-1, two modules would be deemed as outliers, that is, the D_e module values are greater than or equal to one standard deviation above the mean. The designs for **perform_word_count** and **create_log** would be reviewed and possible action may be taken. The design team would then move to the detailed design phase, where the internal workings of the modules would be designed. After completing the tasks involved with this design, the D_i counts would be collected. Once again modules having D_i counts greater than or equal to one standard deviation above the mean would be highlighted. Using the statistics from the above example, only one module would be highlighted. The module **message** is highlighted by

Sample Population

Module Name	D_i	D_e	D(G)
Main	4	5	9
Select_file	7	3	10
Perform_word_count	9	8	17
Word_count	6	2	8
Reset_files	2	2	4
Message	11	6	17
Create_log	6	10	16
Standard Deviation	2.77	2.84	4.74
Mean	6.42	5.14	11.57
Outlier Cutoff	9.19	7.99	16.31

Table 1-1

bD_i, but not by D_e. This suggests to the developers that while the external design is acceptable, the internal design may be too complicated or poorly constructed. The modules highlighted by D_i are often complicated or the exact function has become vague, and redesigning of the architectural and/or detailed design may be warranted.

When the design team is satisfied with the detailed design, they will combine the metric counts to compute D(G). The example above highlights all three modules that were highlighted by the D_e and D_i

metrics. The research team has found that many times D(G) will highlight modules that are not highlighted by the external or internal metrics. This indicates to the developers that the while the modules internal and external designs are borderline, the combination may induce errors later.

What Next

After highlighting the "stress points" in a design, one might ask, "what do I do next?" A review of each such module design is suggested. If after such a review the development team believe a redesign is warranted, the module is redesigned and the metrics recalculated. If a redesign is not necessary, or budget and/or development schedule does not permit it, the team can take actions later in the life-cycle to recognize the potential for problems in these modules. The actions may include assigning the modules to the more skilled members of the implementation team or increasing the amount of testing for these modules.

Chapter 2

Other Detailed Design Metrics

Detailed Design Metrics

Unlike the architectural design metrics, detailed design metrics are not concerned with the context of a module, but only measure the characteristics of the internal design. Such characteristics as size and complexity are the fundamental basis for these metrics. The term "detailed" is used, for the function of the module is not measured, but the details or mechanics of the module are scrutinized.

The History of Detailed Design Metrics

In the early eighties architectural metrics had not been the object of much research, as the process of architectural design development was not fully understood. After the completion of a software project the only tangible item the researchers had to analyze was the code itself. The analysis of this code led to code-level metrics. The development of a methodology for analyzing code was needed, for all other product analyses of a project were subjective. The methodology developed consisted of a post-implementation analysis of the code. The detailed design metrics were used to analyze this code, and after statistical analysis of the measurements, some action was taken.

Detailed Design Metrics

In their article, An Empirical Study of Software Metrics Li and Cheung outlined four types of detailed design metrics. The four groups consisted of 1) volume metrics, 2) control flow metrics, 3) data control metrics, and 4) hybrid metrics. The pages that follow will outline and give the standard metrics for these types.

Detailed Design Metric Example Program

In Appendix A, code for the example program THESIS.C is listed. This program, written in Microsoft C Version 6.0, creates a linked list of employee records read from a file, prints them to standard output, and frees the memory used by the list. Appendix B contains a structure chart and flow graphs for the program and modules. From these items counts for selected metrics presented in this chapter are generated. In the pages that follow, these counts and observations about them will be given.

Volume Metrics

The one attribute that all programs share, regardless of implementation, is size. It is useful to measure the size, for it allows us to characterize the software, without respect to it's application. Volume metrics are

valuable as calculating these metrics is easy after the program is completed. It is the driving force behind many models of software development, and productivity is normally based on size measures (Conte 56).

Volume metrics measure the amount of control flow, data and operational objects, functional units, and/or size of the design. These measurements, in coordination with other resources, can be used to fully understand the scope of a project or module. Table 2-1 contains counts for the volume metrics defined below on the THESIS example program (see above).

Volume Metrics

Metric	Main	Load_records	Print_Records	Free_records	Program
SLOC	16	42	17	11	
Statement Count	8	19	11	4	
Function Count	2	4	2	1	
Unit Count	N/A	N/A	N/A	N/A	4

Table 2-1

SLOC

The SLOC (source lines of code) (*S*) metric count is calculated by counting the number of source lines including comments, executable statements, input/output formats, and compiler directives. There has been some debate on whether the comment lines should be added to the total count, but the most traditional use has been to count every non-

blank line in the code. This metric is not directly applicable in our research, for the development team will not have the finished code until after the implementation phase. A preliminary measure of finished lines of code can be approximated by counting the number of lines in the detailed design.

Note that the professional standard productivity metric KLOC\PM (thousands of lines of code per person/month) is derived from this measure. This measurement is also used in many project schedule approximation models.

Statement Count

A statement count metric *(S_s)* for a module is generated by counting the number of executable statements in the finished code. The type of statement is not important, be it loop, conditional, assignment, function, procedure, or subroutine call, or any other type of atomic operation. This count achieves a better assessment of the actual activity of the module than the SLOC metric. As with the SLOC count, accurate counts cannot be collected until late in the development life-cycle, but approximations can be generated using the detailed design documents.

Unit Count

A unit count *(U_n)* is the sum of the number of modules in a program.

The definition of a module has not been completely agreed upon, so the counts may differ from person to person. The definition most used when counting this metric is that each function / procedure / subroutine is an individual module. Obviously, this metric says little about the complexity or size of the program, but, is computationally inexpensive. This measure can give a feel for the size of a program, but does not seem to have any other significant value.

Halstead's Software Science Metrics

In the late seventies, Halstead noted that some lines of code were harder to implement than others, so the SLOC metric was not always consistent. He felt that another view of programming must be taken to get an accurate account of the volume of a program. With this knowledge he began his research on the Software Science Metrics.

In Halstead's research, he noted that all programs consist of operators and operands, each of which was called a "token". He felt that if the amount and frequency of these tokens could be measured, those counts could give an accurate indication of the volume of the program. These measurements are known as the *Software Science Metrics.*

The software science metrics consists of the four atomic elements:

n_1 = number of unique operators

n_2 = number of unique operands

N_1 = total occurrences of operators

N_2 = total occurrences of operands

An operator is any token that can be used as a relation between two operands. In a programming sense, an operator is any symbol that specifies an action. An operator can be an arithmetic symbol (e.g. +, -, /, *, %, ...), command names (e.g. FOR, WHILE, NEXT, ...), special symbols, (e.g. :=, ==, !=, ...), or some macro or function names (e.g. feof, open, ...).

An operand is any token which is used to represent a data item. All declared data and constants in a program are classified as operands. Functions used as parameters would also be classified as operands.

After collecting these counts, we can determine that volume of a program by the total number of tokens

$$n = n_1 + n_2,$$

and the total implementation length

$$N = N_1 + N_2.$$

Using these metrics, we can find the volume (V) of the program, which is the size of implementation. This can be seen as the number of bytes that the code will need for the implementation. The volume is defined as

$$V = N \; Log \; n.$$

Note that this metric was not aimed at the module, or detailed design

level, but to be calculated on whole programs as individual entities. An example calculation will be given at the end of this chapter. In Table 2-2 the Software Science Metric counts for the thesis program are given.

Halstead's Software Science Metrics

Operators

Number	Type	Number	Type
4	!=	2	char
30	"	1	define
4	#	5	else
6&	,	12	employee_record
44	(	2	employee_type
44	)	1	FILE
38	*	1	fopen
18	,	1	free
10	->	2	free_records
5	.	1	fscanf
26	/	3	height
6	if	2	struct
3	include	1	typedef
3	int	17	}
3	load_file	17	{
43	;	1	main
3	<	1	malloc
14	=	1	memcpy
5	==	2	name
3	>	5	next
3	address	7	void
3	age	3	weight
14	printf	3	while
3	print_records	2	[
2	return	2	]
2	sizeof		

Total Number of Unique Operators = n1 = 55
Total Number of Occurrences = N1 = 459

Operands

Number	Type
5	counter
2	FILENAME
3	fp
4	list_ptr
9	NULL
7	temp
3	temp_ptr

7	top
7	t_ptr
17	x_ptr
1	1
1	20
1	30
3	5

Total Number of Operands = n2 = 14
Total Number of Occurrences = N2 = 70

Tokens = n1 + n2 = 69
Implementation Length = N1 + N2 = 529
Volume = N1 log n1 = 529 log 69

Potential Vocabulary = n* = n1 + n2 >= 2 + n2 = 69 >= 16

Table 2-2

Function Count

According to the widely accepted structured programming methodology, programmers should think of programs as functions, not as modules or groupings of code. A function is defined as a collection of executable statements that performs a certain task. (Conte 43) This Idea of "chunking" (Coulson 43) is the division of modules into one or more functions. The function count metric *F* is calculated by taking the count of these "chunks" or functions within a given design. The count is algorithm-dependant, so the counts for different implementations of the same task will have different function count values.

Flow Control Metrics

Flow control metrics measure the attributes of the logical flow through a module or program. The flow control metrics view code as directed graphs. The counts are collected through the analysis of the graph, characterizing the control complexity of the module.

The flow of control in a computer program normally proceeds sequentially. It is interrupted by three possible situations.

A *forward branch* - This follows a conditional test that leads to a choice of two possible actions.

A *backward branch* - Used to create loops, this may be unconditional (as used at end of a for loop), or may follow a conditional test (as used in a do .. while loop), that allows another iteration or the termination of the loop.

A *horizontal loop* - Typically a transfer of control to a procedure, function, or subroutine, this situation is normally considered an interruption, since the procedure is supposed to return control to the statement following the branch when it terminates. (Conte 62)

In order to fully understand the graph theoretic approach one must understand how the graphs are derived. All directed graphs consist of the two entities:

1) Node A sequential block of code with unique entrance and

exit but no internal branch or loop. (Cheung 700)

2) Edge Flow of control between various nodes. (Cheung 700)

The derivation of a graph from code follows the steps:

1) Identify the nodes - To identify nodes, one finds each block of code that has one entry and one exit, with no internal loops.

2) Connect the nodes - Connect the nodes with the appropriate edges.

Examples of this method can be seen in the generation of the flow graphs given in Appendix B, for the code listed in Appendix A.

In the next several pages the standard control flow metrics will be defined. Table 2-3 contains counts for the flow control metrics on the thesis program.

Flow Control Metrics

Module	V(G)	DC	CL	CALLS	BD+CALLS
Main	2	1	3	3	4
Load_records	6	5	7	0	5
Print_records	3	2	4	0	2
Free_records	2	1	3	0	1

Table 2-3

Decision Count

The simplest flow control metric is the Decision Count *(DC)*, which is the sum of the conditional statements within a module. This metric

gives a accurate account of the volume of conditional flow of a module. The strongest argument against this metric is that it does not tell us anything about the size of a module. This metric, used with volume metrics such as the statement count has been used as a yardstick for module efficiency.

McCabe's Cyclomatic Complexity Metrics

One of the most widely accepted metrics, *McCabe's Cyclometric Complexity metric (V(G))* measures the number of "basic" paths through a program. In more precise terms,

$$V(G) = \text{EDGES} - \text{NODES} + (2 * \text{UNITS}),$$

where EDGES and NODES are defined above and UNITS is the number of connected components within the graph. This measure can also be seen as the sum of the regions in the plane. This measure only encompasses the control structure, without respect to the size or volume of the module.

Gilb's Metrics

Gilb, working with McCabe, proposed several other metrics which would encompass the volume of the module as well as the control structure. The *CL*, absolute logical complexity metric is equal to the total number of binary decisions within the module. After further examination, it was

found that the *V(G)* and *CL* metrics are very closely related. Notice that for any strongly connected graph

$$CL = V(G) + 1.$$

The CL metric still does not account for module volume, so Gilb developed metrics which relate the logical complexity to module size. The cL metric is the ratio of CL to the statement count for a given module. The function

$$cL = CL / S_s$$

gives us an accurate account of the complexity, related to the size of the module. Two other metrics related to Gilb's research are :

CALLS = The total number of calls to other modules

CA + BD = The sum of the calls and binary decisions.

The second metric listed, *CA + BD*, relates module complexity to architectural design. The Central Calls component of the D_i metric is closely related to these ideas and was developed after reviewing McCabe's and Gilb's research.

KNOTS Count

The Knot count (*KNOT*) for a module is the sum of knot occurrences within that module. A *KNOT* count is said to occur when two control transfer intersect. The rationale for this metric is that a module that contains *KNOTS* will be harder to understand than a module without

KNOTS, and thus will be more difficult to implement, test, and is apt to be error-prone.

Reachability and Minimum Number of Path Metrics

Defined by Schneidewind and Hoffman, the reachability and minimum number of paths metrics measure the run time characteristics of the nodes within a module. The minimum number of paths metric *(N_p)* views nodes as unique sequences of arcs. These arcs represent the flow of control of the module to the node, resulting in each node having a N_p count. The N_p count is the fewest number of nodes needed to be visited from entry into the module to the node being analyzed. A related metric is the reachability metric, *R*, which is the sum of the unique ways of reaching the node. When analyzing this count, problems can occur. A module with 50 lines and 25 IF_THEN-ELSE structures will have more than 2^{25} control paths, making the generation of a complete list of the module control flows nearly impossible. To alleviate the problem this presents, the determination of the N_p count excludes the paths with backwards loops traversed more than once. As the R metric can be cumbersome to calculate on large projects, an alternative metric, average reachability metric *(R_)* was suggested by Shooman. This metric is calculated by taking the total number of paths divided by the number of nodes within a module. Table 2-4 contains the thesis

program's reachability and minimum number of paths metric counts.

Reachability and Minimum Number of Paths Metrics

Module Main

Node	Np	R
1	1	1
2	2	1
3	2	1
4	3	2

R_ = 1.25

Module Load_records

Node	Np	R
1	1	1
2	2	1
3	3	21
4	2	20
5	3	20
6	4	20
7	4	20
8	5	20
9	5	20
10	6	20
11	6	20

R_ = 183/11

Module Print_records

Node	Np	R
1	1	1
2	2	1
3	2	2
4	2	2
5	3	3

R_ = 1.8

Module Free_records

Node	Np	R
1	1	2
2	2	2
3	2	2

R_ = 2

Table 2-4

Nesting Levels

In any programming language, nesting structures facilitate the optimization of many algorithms, and are necessary for clear programming. As with all things, too much of a good thing is bad. The use of nesting is necessary, but, a module that has many nesting levels becomes hard to understand, thus difficult to code, test, and maintain. It is this attribute of design that the *Nesting Levels Metric (NL)* measures. Each statement within the program has a nesting level defined by the number of nested blocks[1]. The software engineer may want to get a definitive description of the module nesting level. A accurate nesting level count for the module can be acquired by finding the, *average nesting level (NL_)*. The *NL_* metric is the mean of the nesting levels of the statements within the module.

Data Control Metrics

The fundamental function for any program is to process data, thus it seems prudent to measure the data being processed. The data control metrics measure the scope, size, amount, or complexity of the data being processed within a program or module.

---

[1] Blocks - As used in the compiler theory terminology. A block consists of code that will be executed, any number of times, by the control of a single atomic operation. (e.g. BEGIN .. END (Pascal), { .. } (C), IF .. ENDIF (BASIC))

Variable Count

The simplest data control metric is the *variable count (VARS)*. This metric is calculated by counting the number of variables used within the piece of code being analyzed. It is important not to count the variables defined but not used within the code. To collect this count, it is useful to use the cross-reference table which can be generated by most compilers.

Halstead's Derived Metrics

Halstead's Software Science (N) can be classified as a data control metric, as well as a volume metrics. In his research, Halstead derived several metrics that measured the data control within code. Using the software science metrics we can find the potential vocabulary by using,

$$n^* = n1 + n2 >= 2 + n2.$$

This function is possible because in a minimal form, the number of operators is 2.

Data Usage Metrics

The data usage metrics measure the use and scope of variables with an program or module. The two most accepted data usage metrics are the *live variable* and *variable spans* metrics. The live variable *(LV)* metric

is the count of the number of variables which are deemed "live" for each statement. A variable is considered "live" within a statement if the statement lies on or between the first and last access to that variable. To gain a live variable measurement for a module, the *(LV_)* metric, average live variables with statements, is used. The variable span *(VS)* metric captures the frequency of the use of the variable. The count is the average number of lines between each pair of sequential references of the variable. To gain the module count, the average of *VS* is calculated. Note that this count gives measurements for the variables, not the code. Table 2-5 contains the data control counts for the thesis program.

Data Control and Nesting Level Metrics

Module Main Line #	NL	Live Variables
39	1	
40	1	
41	1	list_ptr
42	1	list_ptr
43	2	list_ptr
44	2	list_ptr
45	2	list_ptr
46	2	list_ptr
47	2	list_ptr
48	1	list_ptr
49	2	list_ptr
50	2	list_ptr
51	2	list_ptr
52	1	list_ptr
53	1	list_ptr
54	1	
55	0	

VARS 1 Variable	Span

```
list_ptr        2

VS_ = 2
LV_ = 13/17
NL_ = 23/17

Module Load_Records
Line #          NL      Live Variables
57              0
58              1
59              1
60              1
61              1
62              1       top
63              1       top, fp
64              2       top, fp
65              2       top, fp, counter
66              2       top, fp, counter
67              3       top, fp, counter
68              3       top, fp, counter, temp
69              3       top, counter, temp
70              4       top, counter, temp
71              4       top, counter, temp, t_ptr
72              5       top, counter, temp, t_ptr
73              5       top, counter, temp, t_ptr
74              5       top, counter, temp, t_ptr
75              5       top, temp, t_ptr
76              4       top, temp, t_ptr
77              5       top, temp, t_ptr
78              5       top, temp, t_ptr
79              5       top, t_ptr
80              6       top, t_ptr
81              6       top, t_ptr
82              6       top, t_ptr
83              6       top, t_ptr
84              5       top, t_ptr
85              6       top, t_ptr
86              6       top, t_ptr
87              6       top, t_ptr
88              6       top
89              5       top
90              4       top
91              3       top
92              2       top
93              1       top
94              2       top
95              2       top
96              2       top
97              1       top
98              0

VARS 5
Variable                Span
fp              5
top             8
```

temp	8
t_ptr	4
counter	1

VS_ = 26/5
LV_ = 80/42
NL_ = 143/42

Module Print_Records

Line #	NL	Live Variables
101	0	x_ptr
102	1	x_ptr
103	2	x_ptr
104	1	x_ptr
105	2	x_ptr
106	3	x_ptr
107	3	x_ptr
108	3	x_ptr
109	3	x_ptr
110	3	x_ptr
111	3	x_ptr
112	3	x_ptr
113	3	x_ptr
114	3	x_ptr
115	1	
116	0	

VARS 1

Variable	Span
x_ptr	6

VS_ = 6
LV_ = 14/16
NL_ = 34/16

Module Free_records

Line #	NL	Live Variables
119	0	x_ptr
120	1	x_ptr
121	1	x_ptr
122	1	x_ptr
123	2	x_ptr, temp_ptr
124	2	x_ptr, temp_ptr
125	2	x_ptr, temp_ptr
126	2	
127	2	
128	0	

VARS 2

Variable	Span
x_ptr	1.5
temp_ptr	2

VS_ = 3.5/2
LV_ = 1

NL_ = 1.3

Table 2-5

Hybrid Detailed Design Metrics

It is impossible to encompass all the nuances of an object with a singular measure. A detailed design is not an exceptions to this rule. With one volume or complexity measurement we can not say very much about the design quality for a given module. It is from this reasoning that the hybrid metrics were developed. A hybrid metric contains several measurements each encompassing the data control, flow control, functional purpose, volume, or any other accessible attribute. In most cases, hybrid metrics proposed use the existing traditional detailed design metrics as components in an equation which will give the metric counts.

Vector Hybrid Metrics

As proposed by Dunsmore, Conte, and Shen the idea of a vector table of metrics would be the most natural way to incorporate two metrics that measure dissimilar attributes. Using this approach we would use metrics A and B, combining them for a hybrid metric (HM) value of $HM(a_i,b_i)$. A vector table would be constructed from existing data for the metric value. This metric value need not be numerical, but could

be descriptive. Such descriptive values might be low complexity, borderline, high complexity. From these counts actions might be taken during the detailed design reviews. One action might be to reevaluate all modules with a HM value of high complexity. Obviously, the central problem with this approach lies in the generation of the descriptive value vector table, which may be somewhat subjective.

Li and Chung's New_1 Metric

Using the fundamentals of software science metrics, Li and Chung have proposed a metric which attempts to measure both volume and control organization. Using such metrics as vocabulary and scope, they have developed an equation that incorporates volume metrics into a purely graph-theoretic approach. For brevity, I have chosen not to include the mechanics of this metric, as its inherent complexity makes it prohibitive.

The D_i Metric[2]

Proposed by the Ball State University Metric Research Team, the D_i metric tries to encompass volume and data control complexity within a given module. This metric is a linear function with three metric counts as the coefficients. These coefficients are:

---

[2] The D_i metric is defined in grater detail in Chapter 1 - D_i Definition and Background.

Central Calls

The central call component is similar to the calls volume metric. A central call is any call to a module which has been developed for the current project.

Data Structure Manipulations

The data structure manipulations count is classified as a data control metric. This metric stems from the operators metric used in the software science metrics. A data structure manipulation is any indirect access to a data item.

I/O Count

The I/O count stems from the volume metric I/O Formats (IOFORMTS). An I/O count is any functional access to a external device, such as a monitor.

After collecting these metrics for a module, the counts are applied in the equation

$$D_i = w_1(CC) + w_2(DSM) + w_3(I/O)$$

to find the metric count for the module. Having stemmed from the standard metrics, D_i apparently gets a accurate account of the module's propensity for error. Table 2-6 contains the D_i counts for the thesis example program.

Di Metric Counts

Module	CC	DSM	I/O	Di(1)	Di(2)
Main	3	4	4	11	17
Load_records	0	11	4	15	31.5
Print_records	0	8	8	16	28
Free_records	0	4	0	4	10

Table 2-6

Chapter 3

A Pilot Study of CS680 Projects

The Pilot Study

In January of 1990 the metric research team at Ball State University began a pilot study on design metrics. Each fall at Ball State University, a graduate level software engineering class is taught. In this class, teams of students develop a software package using the waterfall software development model. In the interest of verifying the D_e, D_i, and D(G) design quality metrics, the research team concentrated our efforts on projects from the software engineering class of Fall 1990. The study was conducted from January 1990 to April 1990.

The projects selected presented us with a reasonable test bed, for we had the documentation and code of each system for evaluation. The students deliver documentation from all phases of software development, so that error tracking is possible. In addition to all these tangible items, we had the ability to interview the developers about the selection

of certain designs over others. This ability to interact with the developers "after the fact" allowed us to clearly understand all aspects of the system designs.

The metrics team felt it prudent to look at other traditional metrics to ensure our research was both unique and valuable. While testing our initial D_i metric, we felt it necessary to compare its results with McCabe's cyclomatic complexity, V(G), metric. This is valuable, as it can be calculated at the detailed design phase. Other measures, such as SLOC, source lines of code, can only be calculated late in the development process, but is so widely used as a standard metric, it too was selected to be tested.

Objective

By applying our metrics to the projects, we hoped to verify the D(G) metric as a good indicator of design quality by highlighting error-prone modules in the systems. In the months prior to this study, the D_i metric was developed by the design metrics team at Ball State University. The team had debated the theoretical foundations of the metric and was ready to prove (or disprove) the suppositions and expectations of their research. The D_e metric was developed some time before, as outlined in chapter 1. At most, the D_e metric was tested on

a small test bed, and further verification was warranted. The availability of resources needed to calculate and interpret the metrics was cited as the central reason for using this data set. Great care was taken in the collection of data, so as to ensure the integrity of the study.

The Project

The project assigned to the software engineering class required the students to design and implement a generic employee database system. The project was designed to allow for a wide interpretation of the requirements, thus giving a wide array of designs to analyze. The development of the system was assigned on September of 1989 and was finished on or about December 20, 1989.

In addition to the actual finished system, the students were required to submit all design documents. It was from these documents, after verifying the adherence of the finished systems to these documents, that the metrics were collected.

The exact requirement specifications that were given to the students are listed in Appendix C - Pilot Study Project Specifications.

The Project Teams

The teams of developers consisted of two to three individuals with varying experience and knowledge. The students experience ranged from several years of professional experience to first semester computer science students. In all, the class contained 15 students constituting six project teams.

The class was held for seventeen calendar weeks. The students were graded on several tests and the ability of the teams to complete all phases of the development life-cycle adequately.

The Finished Projects

All three projects selected to be used in this study were completed and demonstrated in the last week of the semester. A representative from each group was selected, and this representative demonstrated each major requirement of the project. These demonstrations helped some members of the metrics team become familiar with the systems.

It is interesting to note that the structure charts for the projects differed greatly. The requirements were interpreted similarly by all teams, but the resulting designs differed greatly.

Appendix D contains the actual D_i counts from the projects.

Results

The statistical results of the pilot study were very encouraging. We found that our D_i metric performed better than any other metric at locating "stress points" in a detailed design.

LOC (over 50) as a Predictor of Error-Prone Modules	
Modules Highlighted	15%
Highlighted Modules with Errors	50%
Errors Found	61%
Error Modules not Found	50%
Errorless Modules Highlighted	50%

Table 3-2

The team felt it necessary to check our metric against traditional metrics. In Table 3-2 we show the effectiveness of SLOC, or source lines of code, as a design quality metric. Highlighting 50% of the modules, and only finding 61% of the errors, SLOC > 50 does not seem to have fared well.

The second metric we tested was McCabe's cyclomatic complexity, or V(G), on our test bed. This metric is designed to detect the complexity

In Table 3-1, the specific attributes of the projects used in the study are listed.

Pilot Study Projects

Team	Language	SLOC	Number of Modules	Total # of Errors	Number of Error Modules
1	Pascal	478	17	34	4
2	Pascal	659	21	14	2
3	C	445	33	12	4

Table 3-1

The Data Collection Process

The metrics were collected by the members of the design metrics research team in January of 1990. All counts were calculated manually, and verified by other members of the metric team. The D_e counts were collected from the project structure charts. The D_i counts were collected from the actual code, which was submitted with the final projects. After all data was collected and verified, the D(G) counts were calculated from the original D_e and D_i counts. The counts were then distributed to the members of the team to be analyzed.

Appendix D contains the actual D_i counts from the projects.

Results

The statistical results of the pilot study were very encouraging. We found that our D_i metric performed better than any other metric at locating "stress points" in a detailed design.

LOC (over 50) as a Predictor of Error-Prone Modules	
Modules Highlighted	15%
Highlighted Modules with Errors	50%
Errors Found	61%
Error Modules not Found	50%
Errorless Modules Highlighted	50%

Table 3-2

The team felt it necessary to check our metric against traditional metrics. In Table 3-2 we show the effectiveness of SLOC, or source lines of code, as a design quality metric. Highlighting 50% of the modules, and only finding 61% of the errors, SLOC > 50 does not seem to have fared well.

The second metric we tested was McCabe's cyclomatic complexity, or V(G), on our test bed. This metric is designed to detect the complexity

V(G) (>= 10) as a Predictor of Error-Prone Modules	
Modules Highlighted	11%
Highlighted Modules with Errors	44%
Errors Found	37%
Error Modules not Found	66%
Errorless Modules Highlighted	56%

Table 3-3

of a given module. In Table 3-3, we find that V(G) found 37% of the errors, highlighting 11% of the modules.

D_i (unit weights) as a Predictor of Error-Prone Modules	
Modules Highlighted	9%
Highlighted Modules with Errors	86%
Errors Found	62%
Error Modules not Found	50%
Errorless Modules Highlighted	40%

Table 3-4

In testing our D_i metric, we endeavoured to calibrate the w_n weights. Using unit weights[1], D_i highlighted 9% of the modules, which contained 62% of the errors in the system. Using this scheme we incurred a 14% false positive rate, which in comparison to 56% of false positive rate of

[1] unit weights - $w_1 = w_2 = w_3 = 1$

the SLOC metric, was more than acceptable. The results of this test are listed in Table 3-4.

The correlation between errors and data-structure manipulations was significantly higher, thus the new weighting scheme was generated. After viewing the data, we decided to try the weighting scheme of $w_1 = w_3 = 1$, and $w_2 = 2.5$, thus stressing the data structure manipulations. The resulting formula

$$D_i = CC + 2.5(DSM) + I/O$$

is used. Using this weighting scheme we found 92% of the error in the projects while only highlighting only 8% of the modules. Note that **no** modules were highlighted that contained 0 errors. These results are listed in Table 3-5.

D_i (W_2 = 2.5) as a Predictor of Error-Prone Modules	
Modules Highlighted	8%
Highlighted Modules with Errors	100%
Errors Found	92%
Error Modules not Found	40%
Errorless Modules Highlighted	0%

Table 3-5

Conclusions

D_i, at least for some small projects, is successful at determining quality of detailed designs. After calibrating the weights, D_i highlighted modules which contained more than 90% of the system errors. This kind of result prompts this research team to collect more data. We have found such an unbelievably high success rate, that more verification is warranted.

It has been proposed that a acceptable criteria for a good metric would find 80% of the errors by highlighting 20% of the modules in a system. In this study, we have bettered this mark by finding 92% of the errors by highlighting only 8% of the modules. If this metric could maintain these kinds of results on large scale software, the value of the application of these metrics to professional environments would be undeniable.

The traditional metrics SLOC and V(G) were tested on the pilot study projects and did not seem to return any significant results. SLOC managed to highlight modules which contained 61% of the errors, but in doing so created a 50% false positive rate. In addition to this the correlation between module errors and SLOC was insignificant. The V(G) metric highlighted modules which only contained 37% of the

modules. In any case, neither metric does not seem to hold up as a reasonable predictor of design quality when scrutinized.

Having found these results on small scale software, it would seem reasonable to ask, "Sure, the metrics seem to have better results than most, but, will it perform the same in a professional environment?" This question will be answered by the remainder of this thesis.

Chapter 4

The Data Collection Process

When developing large-scale software it does not seem reasonable to hand calculate the D_i metric counts for each module. Realizing the need for a automated metric calculation tool, the Ball State University metrics research team developed a Software Detailed Design Analyzer (SDDA). This system is intended to generate detailed design metric counts from the documents created in the detailed design phase. In this chapter I will list the assumptions underlying this tool and outline the mechanics of the system. The system design was developed for the specific needs of this research, and with some modification will be able to collect counts in a industrial environment.

The Software Design Analyzer

Developed by the design metrics research team of Ball State University, the Software Design Analyzer (SDA) calculates the D_e architectural metric from design documents developed in the architectural design

phase. The input documents, developed in a CASE[1] tool, are evaluated and metrics are collected. Using this tool, one could detect the status of a project under development during or at the end of the architechtural design phase. As the use of software engineering methodologies becomes more prevalent in today's development facilities, the need for greater support of metrics would seem to make sense. The SDDA tool is the counterpart to this system, as they both calculate metrics during the design phase.

The Data

The data, supplied by the Computer Science Corporation, consisted of 25,000 lines of code written in the ADA programming language. The code encompassed 21 programs which are part of the STANFINS project, a 2.5 million line system developed over the years 1986 - 1990. The code was contained in 150 code files, which included approximately 2180 modules[2]. The project target platform was the VMS/VAX DEC Environment. As all support libraries were not available to the research team, the resolution of some data was not possible. While this resolution was not possible through automated avenues, all cases were

---

[1] CASE - Computer Aided Software Engineering. A case tool provides project development and management utilities.

[2] The definition of a "module" used in the calculation of the metrics was any procedure, function, task or package which had an accessible code body.

dealt with individually and judgements were made by the researchers. This not withstanding, consistency of data was scrupulously checked.

Objectives

The objectives for the detailed design analyzer consisted of 1) to accurately calculate the D_i counts on the ADA code, 2) report these counts in a format which would facilitate the statistical analyses of these counts, and 3) provide for future enhancements. With these objectives in mind, a model that would be reliable, flexible, and coherent was designed (see The SDDA Pipeline below). After analyzing the resulting SDDA system, I feel these requirements have been adequately met.

The SDDA Pipeline

In analyzing the SDDA functional needs, a series of distinct processes of data calculation and analyses became apparent. This series of processes became the model for the software design. This model has been named the SDDA pipeline. In Figure 4-1, a diagram of this pipeline is given. The three central operations of the SDDA are defined as:

1) **Calculation of the Metrics** - The calculation of the metrics from the design documents is the most important process in

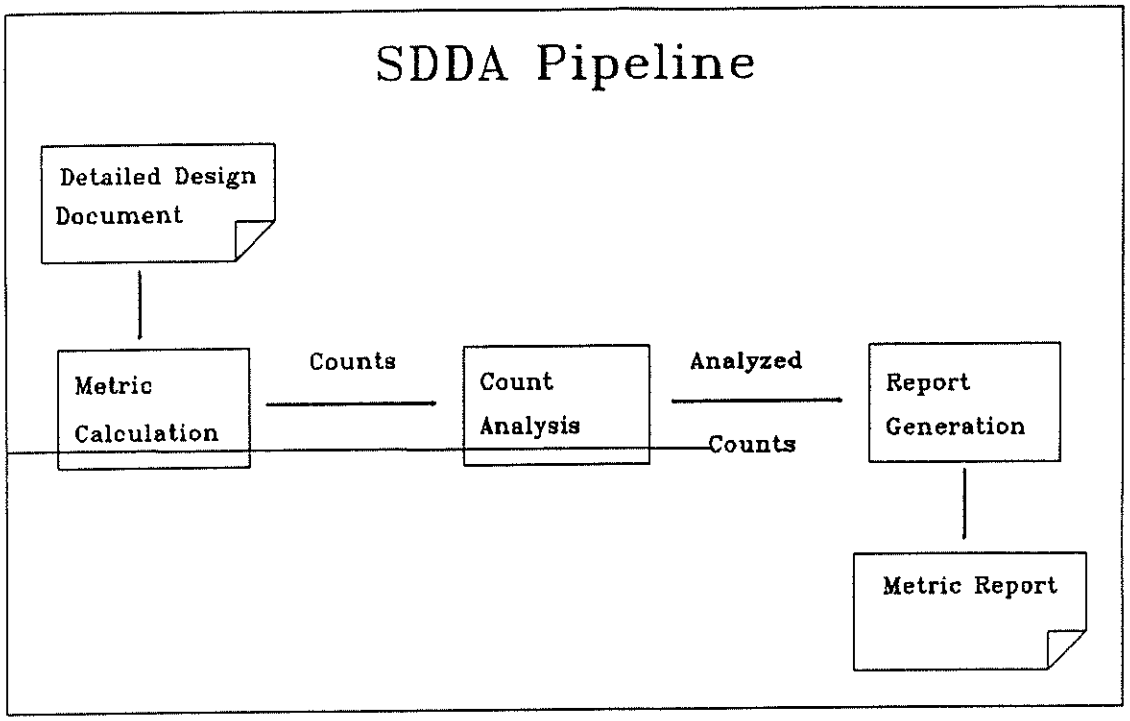

Figure 4-1

In the scope of this thesis we have parsed the ADA code, and using the information included in these doucments, calculated the metrics. The counts were calculated for each module in the 21 programs, and output a file which was intended as input for the SPSS/PC+ statistical package.

2) **Analyze the Counts** - As the counts are meaningless in and upon themselves, the statistical analysis of metrics is needed. The count population needs to be analyzed so that outliers can be identified, and future action may be taken. Such packages as SPSS and SAS can be used, as well as user derived applications for this analysis. For this thesis

outliers can be identified, and future action may be taken. Such packages as SPSS and SAS can be used, as well as user derived applications for this analysis. For this thesis I chose to use the SPSS/PC+ software package to analyze the counts resulting from the calculation.

3) **Report the Counts** - The reporting facility does not really add any new non-cosmetic functionality to the tool, but rather allows a quick and less cumbersome review of the resulting output, as well as management of the counts. In many cases, a project manager may not want to look at the individual counts for thousands of modules. In this case a report of the "big picture" would be in order. Due to time constraints, I was not able to add in this functionality, other than the standard reporting facilities of the SPSS/PC+ package with this research. In addition to generating reports, this application might keep the counts in a database for future use.

During the development of this tool, it became obvious that three separate applications should be developed. The original design contained one application encompassing all three processes, and was deemed too complicated. The resulting multi-application design is clearer and in the end analysis, easier to implement.

Implementation Issues

Having the greatest availability to the research team, we selected the VAX 11/780 unix platform to develop the SDDA, but have plans to port these applications to the SUN workstation platform at a later date. The Mt XINU UNIX operating system is used on the computer science VAX. The unix operating system's multi-processing capability became instrumental in the application's design, for the ability of the three applications working in tandem facilitated the final design (see above). Speed and work space never became an issue during development, but for smaller systems, one might optimize the code further.

The Implementation

The next several pages will outline the actual design and implementation of the SDDA software. For brevity, alternate designs and reasoning for design decisions will not be included.

The Metric Count Calculator

Central to the problem is the actual semantic analysis of the detailed design documents. The job the metric count calculator is to parse the documents, glean the semantic meaning of the objects within the

document and accurately count the D_i element occurrences. In this research the detailed design documents consisted of the actual finished ADA code. It seems prudent to take the symbol tables for an actual compile of the ADA code and resolve the occurrences of data and functional accesses. Using this logic the design for the metric calculator was developed, and the decision to use a pre-compiler to interpret the semantic meanings of the design documents was made.

Two language development tools were used to generate the count calculator. The unix LEX (Lexical Analyzer Generator) and YACC (Yet Another Compiler-Compiler) were used to generate the lexical and syntactic parsers. The LEX application accepts a lexical grammar and generates lexical parser C code. The YACC application accepts a EBNF[3] grammar and generates syntactic parser C code.

The parsers generated are actual C code with hooks to attach user defined C routines. Using the code generated by the utilities along with code written by the research team the calculator was compiled. The job of the development team was to generate a symbol table and occurrence resolution routines.

The final design resulted in a two pass compiler which would parse the code, finding all possible D_i occurrences and generating a symbol table

---

[3] EBNF - Extended Backus - Naur Form - Symbolic form used to specify a complex set of rules.

on the first pass. The second pass would resolve the possible occurrences and calculate each module's counts. The data used in this thesis required that symbol tables be generated over several files, as the programs defined and accessed data over several files. As information to resolve some of the data was not present, the members of the research team resolved those through external documentation and common sense.

The Count Analyzer

The count analyzer's job is to take to raw counts, calculate D_i, and identify the outliers. More analysis may be warranted, such as the calibration of k_n weights. The application may use an existing package or the user might want to develop a specific program. The advantage of a user generated application is in that interface to output of the metric calculator and the and input the report generator are easily modified. For this thesis the SPSS/PC+ statistical analysis package was used. Due to the exstensive statistical tests needed to test the behavior of the metrics, the functionality of SPSS/PC+ was needed.

The Report Generator

The purpose of the report generator is to take the metric counts and put them into an acceptable form. A user may want to generate

reports at several times, or view the counts over time. This application might also keep the counts in a data-base. The development of a project could be tracked from the beginning to the end of the detail design phase. These utilities are not really part of the count analyzer, but would be useful in a professional environment. For this thesis, the standard SPSS/PC+ reports were utilized.

Chapter 5

Results of STANFINS Data

Introduction

This chapter will outline and explain the results of statistical tests performed on the data collected from the STANFINS project. The method for the data collection is given in Chapter 4 - The Data Collection process. Two approaches were taken to analyze the data. The first approach is a result-oriented scheme. This scheme actually applies the process of finding cutoff points and highlighting modules accordingly. Using these modules, we determine statistics such as hit-miss ratio, number and percentage of errors found, error modules found and errorless modules highlighted. With these data, we can determine the effectiveness of the metric in practice. In addition to the standard process, the X-Less algorithm, developed by the Ball State Metric Research Team, is presented and used on the STANFINS data. The second approach is a cause-oriented scheme. This scheme uses regression analysis techniques to verify if the metric is actually a predictor of error prone modules, as well as approximating the chance of the results occurring by chance. These statistical tests will give us a better look at the cause-effect relationship between D_i and the existence of error-prone modules. Finally, observations, explanations,

and areas of future study will be given.

Limitations of The Data

The data collected from the STANFINS project contains some problems over which the researchers had no control. First and foremost is the absence of any I/O counts in the data. This was due to the structure of the ADA language, which has no standard I/O functions. In ADA, all I/O must be handled in library modules and accessed through packages. The I/O package which has been used throughout all 21 programs was not available, and the I/O functions used were not defined as such, so any assumptions made by the research team about the I/O would make the data suspect. Secondly the amount of data structure manipulations outweighs the central calls greatly. For the 21 program analyzed, the STANFINS project consisted mainly of database records manipulation. In most cases the modules consisted of the initialization, field manipulation, and verification of database records. This led to the massive counts for data structure manipulation. It is the belief of this researcher that while other types of programs would result in significant variation in count distribution, the metric would still preform well. These problems not withstanding, the data resulted in meaningful results.

A Result-Oriented Approach to Data Analysis

It is necessary to analyze the data as it would be used in a real development environment, that is, if the developers of the system used the D_i metric, would they have benefitted from its use? The statistics gathered will determine the effectiveness of the metric as used, removed from the meaning of the effectiveness.

The programs were divided into three groups by CSC personnel. Each of the three groups, named easy, medium, and hard, consisted of seven programs of varying sizes. These groups were determined by the difficulty of the program. The criteria which would determine grouping encompassed the function, design, and implementation of the program. Obviously, a program with simple function, design, and implementation would fall into the easy group, medium in the medium group, and hard into the hard group. While it is suspected that the program measures were subjective, viewing the code themselves leant a measure of validity to the groupings. Appendix E lists the actual metrics counts for each module in the three groups.

The Easy Group

The easy group of programs contains usually small programs, which perform functions which are easy to design and implement. Table 5-

1 contains statistical information about the analysis of D_i on these programs. The table uses the unit weight weighting scheme, ($w_1 = w_2 = w_3 = 1$), for the analysis.

Easy Group Statistics 1

Di Weighting Scheme
w1 = 1
w2 = 1
w3 = 1

Cases	=	163
Mean	=	37.073620
Stdev	=	44.764895
Cutoff	=	81.838514
Sum	=	6043.0000

Type		Sum	Percentage
Modules	=	163	N/A
Total Errors	=	175	N/A
Error Modules	=	31	19.02%
Modules Highlighted	=	18	11.04%
Highlighted Modules with Errors	=	13	72.22%
Errors Found	=	129	73.71%
Error Modules not found	=	18	58.06%
Errorless Modules Highlighted	=	5	27.78%

Table 5-1

The distribution of errors over these programs is normal, as all the errors lie in 19% of the modules. This metric found 73% of the errors in the system, with a false positive rate of 27%, which means 27% of the modules highlighted had no errors. The problem that seems most prevalent is that almost 60% of the error modules were not found. This tells us that 73% of the errors were contained in 40% of the modules.

Easy Group Statistics 2

Weighting Scheme
w1 = 1
w2 = 2.5
w3 = 1

Cases = 163
Mean = 91.067485
Stdev = 111.490803
Cutoff = 202.558288
Sum = 14844.000000

Type		Sum	Percentage
Modules	=	163	N/A
Total Errors	=	175	N/A
Error Modules	=	31	19.02%
Modules Highlighted	=	18	11.04%
Highlighted Modules with Errors	=	13	72.22%
Errors Found	=	129	73.71%
Error Modules not found	=	18	58.06%
Errorless Modules Highlighted	=	5	27.78%

Table 5-2

The weighting scheme where DSM occurrences are weighted 2.5 does greatly effect the performance of the metric; in fact the same modules are highlighted as in the unit weighting scheme. This is due to the great number of data structure manipulations as compared to the small number of central calls.

Overall, the D_i metric performed well on these programs. If used during the detailed design process, potentially 73% of the errors could have been avoided.

The Medium Group

The medium group consists of programs of average difficulty. The size of these programs, in modules and LOC, is larger than for the easy programs. The functionality of the programs do not seem to be much different. In Tables 5-3 and 5-4, statistics for the D_i metric over these programs is given. Table 5-3 used the unit weighting scheme, whereas Table 5-4 uses a 2.5 value for k_2.

Medium Group Statistics 1

Di Weighting Scheme
w1 = 1
w2 = 1
w3 = 1

Cases = 273
Mean = 29.641026
Stdev = 39.067853
Cutoff = 68.708879
Sum = 8092.000000

Type		Sum	Percentage
Modules	=	273	N/A
Total Errors	=	674	N/A
Error Modules	=	67	24.54%
Modules Highlighted	=	34	12.45%
Highlighted Modules with Errors	=	23	67.65%
Errors Found	=	423	62.76%
Error Modules not found	=	44	65.67%
Errorless Modules Highlighted	=	11	32.35%

Table 5-3

Medium Group Statistics 2

Di Weighting Scheme
w1 = 1

w2	=	2.5
w3	=	1
Cases	=	273
Mean	=	72.736264
Stdev	=	96.980687
Cutoff	=	169.716951
Sum	=	19857.000000

Error Analysis Type		Sum	Percentage
Modules	=	273	N/A
Total Errors	=	674	N/A
Error Modules	=	67	24.54%
Modules Highlighted	=	33	12.09%
Highlighted Modules with Errors	=	23	69.70%
Errors Found	=	423	62.76%
Error Modules not found	=	44	65.67%
Errorless Modules Highlighted	=	10	30.30%

Table 5-4

In Table 5-4 we find that the highlighted modules located 62% of the modules by highlighting only 12% of the modules. Notice that the metric did not perform as well as it did on the easy modules. Moreover, the false positive rate increased to 30%.

While it may be said that the metric did not perform as well as on other projects, its ability to locate error-prone modules is unquestionable.

The Hard Group

The hard group encompassed the most difficult as well as the largest programs and modules in the study. On average, the number of modules per program was much larger than in the other two groups.

Tables 5-5 and 5-6 list statistics for the hard group programs using the unit and k_2 = 2.5 weighting schemes, respectively.

Hard Group Statistics 1

Di Weighting Scheme
w1 = 1
w2 = 1
w3 = 1

Cases = 422
Mean = 39.872038
Stdev = 62.540535
Cutoff = 102.412573
Sum = 16826.000000

Error Analysis

Type		Sum	Percentage
Modules	=	422	N/A
Total Errors	=	1398	N/A
Error Modules	=	122	28.91%
Modules Highlighted	=	42	9.95%
Highlighted Modules with Errors	=	35	83.33%
Errors Found	=	720	51.50%
Error Modules not found	=	87	71.31%
Errorless Modules Highlighted	=	7	16.67%

Table 5-5

Hard Group Statistics 2

Di Weighting Scheme
w1 = 1
w2 = 2.5
w3 = 1

Cases = 422
Mean = 98.331754
Stdev = 155.605305
Cutoff = 253.937058
Sum = 41496.000000

Error Analysis

Type		Sum	Percentage
Modules	=	422	N/A

Total Errors	=	1398	N/A
Error Modules	=	122	28.91%
Modules Highlighted	=	42	9.95%
Highlighted Modules with Errors	=	35	83.33%
Errors Found	=	720	51.50%
Error Modules not found	=	87	71.31%
Errorless Modules Highlighted	=	7	16.67%

Table 5-6

The hard modules found 51% of the errors by highlighting 9% of the modules. The metric performed well on these programs, because a small number of modules were highlighted and more than half of the errors were found. The false positive rate for the hard modules was only 16%, meaning that a small number of modules were highlighted that contained no errors. Using these numbers for the hard modules, we find that if the guidelines for the metric were used, potentially the number of errors after the implementation phase would be greatly reduced.

All Groups

The next study covered all modules in each of the three groups. The modules were all weighted equally and no distinction was given to the type of program analyzed. Tables 5-7 and 5-8 give statistics for all of the groups analyzed together, where the weighting schemes are unit weights and $k_2 = 2.5$ in Tables 5-7 and 5-8, respectively.

All Statistics 1

Di Weighting Scheme
w1 = 1
w2 = 1
w3 = 1

Cases = 858
Mean = 36.085082
Stdev = 52.963482
Cutoff = 89.048564
Sum = 30961.000000

Error Analysis

Type		Sum	Percentage
Modules	=	858	N/A
Total Errors	=	2247	N/A
Error Modules	=	220	25.64%
Modules Highlighted	=	87	10.14%
Highlighted Modules with Errors	=	67	77.01%
Errors Found	=	1216	54.12%
Error Modules not found	=	153	69.55%
Errorless Modules Highlighted	=	20	22.99%

Table 5-7

All Statistics 2

Di Weighting Scheme
w1 = 1
w2 = 2.5
w3 = 1

Cases = 858
Mean = 88.807692
Stdev = 131.747182
Cutoff = 220.554875
Sum = 76197.000000

Error Analysis

Type		Sum	Percentage
Modules	=	858	N/A
Total Errors	=	2247	N/A
Error Modules	=	220	25.64%
Modules Highlighted	=	87	10.14%
Highlighted Modules with Errors	=	68	78.16%
Errors Found	=	1218	54.21%
Error Modules not found	=	152	69.09%

Errorless Modules Highlighted	=	19	21.84%

Table 5-8

Analyzing all modules together, the D_i found 54% of the errors by highlighting 10% of the modules. The false positive rate for this test was 21%, meaning for every 5 modules highlighted, approximately 4 contain errors. Considering that one out of every four modules contained errors, we can say that by highlighting 10% of the modules and finding 54% of the errors, the metric performed well. A more detailed analysis of these numbers is given in a later section of this chapter.

The X-Less Algorithm

In the research performed by the metrics research team at Ball State University, we found that in most designs there are large modules that outweigh all others. Figure 5-1 gives a pictorial representation for these modules. Several reasons are sighted to account for these modules. First lies in the scheduling process for most projects. In many cases, design

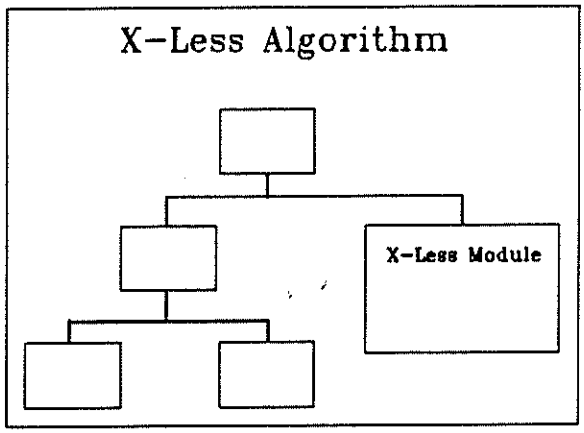

Figure 1

teams find themselves getting close to deadlines for designs. When this happens, they take much of the functionality left to implement and place it in several modules. These resulting modules are large, normally highly coupled, and in general, poorly designed. Another possible reason for these modules lies in incomplete or unusual specifications. If specifications for a program involve many unrelated functions that do not warrant their own modules, designers many design modules that are receptacles for this functionality. If many unrelated task are placed together within a module, it will have low cohesion. It is generally an accepted fact that modules that are not cohesive are error-prone (Conte 108). The result of measuring these modules is that the sensitivity of the metric is reduced. The standard deviation and mean for the metric would be higher, in some cases by orders of magnitude. We have found that be removing these modules from the population, we can find many more modules without sacrificing accuracy. The term "X-Less" means the removal of X percent of the modules from the population. Table 5-9 shows the logic for this algorithm.

The X-Less Algorithm

```
Begin X-Less
        Calculate Metrics
        Complete Statistical Tests
        While (not satisfied)
                Begin
                        Remove X% of modules from population
                        Complete Statistical Tests
                        Increment X
                End
```

Table 5-9

Using the X-Less algorithm, we found the results listed in Table 5-10. All groups and the k_2 = 2.5 weighting scheme were used.

X-Less Algorithm Results

Di Weighting Scheme
w1 = 1.00
w2 = 2.50
w3 = 1.00

Type		Sum	Percentage
Modules	=	858	N/A
Total Errors	=	2247	N/A
Error Modules	=	220	25.64%

Percent Removed	Modules Removed	Modules Highlighted	Highlighted Modules with Errors	Errors Found	Error Modules not found	Errorless Modules Highlighted
0	0	10.14%	78.16%	54.21%	69.09%	21.84%
1	8	13.88%	69.09%	61.37%	64.15%	28.81%
2	17	15.58%	67.54%	64.80%	62.44%	29.77%
3	25	18.73%	58.78%	68.49%	61.11%	36.54%
4	34	20.27%	55.64%	71.61%	60.85%	37.13%
5	42	22.06%	55.07%	74.28%	58.47%	37.22%
6	51	23.92%	53.52%	75.92%	56.82%	37.82%
7	60	25.56%	51.39%	77.97%	56.21%	38.73%
8	68	26.33%	50.71%	78.15%	56.71%	38.94%
9	77	27.02%	47.01%	78.15%	59.62%	39.81%
10	85	27.43%	48.03%	78.37%	60.13%	39.62%

Table 5-10

Table 5-10 shows the results of the X less algorithm on the sample data. The most striking thing about these results is that by removing 5% of the modules from the population one get a 50% increase in the

number of errors found. This is strong evidence that this algorithm should be applied when using the D_i metric. According to these data, the law of diminishing returns applies to this algorithm. After removing 5% of the modules, removing more does not give any significant benefit. The number of modules highlighted does not increase much, so the effect of the module removal is nominal, at best. The reason for this phenomenon is in the distribution of the module counts. After removing 5% of the modules, the range of population counts is reduced dramatically, so outliers from that distribution are few and far between.

Cause Oriented Analysis

It seems prudent to analyze the cause-effect relationship between D_i and errors. This is necessary, for any study that looks at data without regard to its interrelationships is suspect. An accepted way of finding and verifying these relationships is regression analysis. Regression analysis, " ... is a good way to describe and summarize the linear relationship between two variables"(NORIUS 341). In this study we are using multiple regression which assesses the relationships between the dependant variable and two or more independent variables. Some assumptions are made when using this process.

The most important assumption that is made in regression analysis is that the relationship has some theoretic base. One could not perform

regression analysis on the relationship between the type of car ones drives and the high school attended, for this relationship has little intuitive basis. However, a relationship between the type of car one drives and the annual salary can be performed, for this relationship has a reasonable theoretical base. In the latter case, a regression analysis would give evidence for or against this theorized relationship. Moreover, a regression analysis specifies the nature of the relationship through regression coefficient which describes how changes in the independent variables bring about changes in the dependant variable. It is important to note that the analysis does not take into account any problems in the data collection process. If the collection process is sound, then the regression analysis will be sound. In the case of this thesis, every step possible to ensure the integrity of the data was taken. The purpose of the regression analysis of the data is twofold. The first is to determine the existence of the relationship between the D_i metric and errors. Secondly, the analysis will give the optimal error approximation coefficients for the D_i formula. In the case of the D_i multiple regression analysis, the dependant variable is errors and the independent variables are central calls, data structure manipulations, and input/output, for the supposition is that the D_i metric is an accurate predictor of error-prone modules.

The D_i - Error Relationship

The first step in verifying D_i as an error-prone module locator is to determine that the relationship between the two actually exists. In table 5-11, a portion of the SPSS/PC+ output of the multiple regression analysis is given.

Total Modules Analysis

Number of Valid Observations (Listwise) = 855.00

Variable	Mean	Std Dev	Minimum	Maximum	N	Label
CC	.71	1.54	0	14	855	Central Calls
DSM	35.48	52.56	0	653	855	Data Structure Manipulation
IO	.01	.01	0	6	855	Input-Output
ERR	2.62	8.24	0	84	855	Number of Errors

Page 3 Total Modules Analysis

This procedure was completed at 19:41:04
CORRELATION VARIABLES ALL.

Page 4 Total Modules Analysis

Correlations:	CC	DSM	IO	ERR
CC	1.0000	.2913**	-.0157	.1102**
DSM	.2913**	1.0000	-.0225	.5175**
IO	-.0157	-.0225	1.0000	-.0109
ERR	.1102**	.5175**	-.0109	1.0000

N of cases: 855 1-tailed Signif: * - .01 ** - .001

" . " is printed if a coefficient cannot be computed

Table 5-11

The top portion of Table 5-11 gives the raw statistics. The bottom section characterizes the relationships between the elements of D_i and errors, as well as the interrelationships between the elements themselves. The numbers in the columns are the correlation coefficients. These coefficients give the interdependence of one variable's variance with another. In more precise terms, the square of the correlation

coefficient gives the amount, in percent, of the variables' variance due to each other. In the case of the DSM - Error relationship, the correlation is .5175, which squared is .2601, or 26%. This means that 26% of the variance of errors is due to the variance in DSM. Using this formula, we can say that the central calls count is significantly correlated with errors, as is DSM. When looking at these numbers we can also tell that the central calls is significantly correlated with DSM. This brings us to the fact that there is some overlap between the variance attributed to central calls and data structure manipulations. Since the only input/output count involved in the study was due to a data error, we can't say anything about its relationship to errors. Overall this study tells us that DSM is an accurate approximator of errors, and further study is warranted.

The D_i Formula Coefficients

In addition to determining the existence of a valid relationship, regression analysis can find an optimal coefficient scheme for the element formula. Table 5-12 gives a portion of the SPSS/PC+ output file for the regression analysis.

Regression Analysis of Coefficients

---------- Variables in the Equation ----------

Variable	B	SE B	Beta	T	Sig T
DSM	.08116	4.59472E-03	.51751	17.664	.0000
(Constant)	-.25939	.29124	-.891	.3734	

---------- Variables not in the Equation ----------

Variable	Beta In	Partial	Min Toler	T	Sig T
CC	-.04424	-.04946	.91517	-1.445	.1487
IO	7.4073E-04	.00087	.99950	.025	.9799

End Block Number 1 PIN = .050 Limits reached.

Table 5-12

This table gives us an analysis of the "best - fit" coefficients for the data. Using these coefficients, we get the best approximation of errors on this data. Notice that the input/output and central calls elements are left out of the formula. This is because, for this data, they do not add any significant ability in highlighting error-prone modules. This analysis gives us the error approximation formula,

Errors = -.25939 + (DSM * .08116).

This formula is in a slope - intercept form, where -.25939 is the axis intercept and the .08116 is the slope of the line defining errors as a function of the DSM count. This formula not only approximates the existence of errors within a module, but the actual number of errors within the modules.

Conclusions

The most noticeable conclusion that one can draw from this study is that data structure manipulations seems to be a good indicator of software errors. If the nature of the data is not unique to this project, but is normal in all software development, we can say with authority that when a module has a high data structure manipulation count, the module has a very good chance of containing errors. The problem with the study is in that the data collected may not be "normal". There were many extenuating circumstances which may have effected the data. First and foremost is that the input and output functions in the programs could not be determined, for the resources were not at my disposal. Secondly, the programs were database intensive, which in actuality meant the amount of data being manipulated outweighed the control transfer as to make central calls counts insignificant. The result of these two factors made the study only effective on the data structure manipulations count in approximating errors. These factors not withstanding, no documentation was available with the programs, so making observations about the functionality is pure speculation.

Future Research

The results of this analysis lead us to the conclusion that data structure

manipulations are valid approximators of error, but not enough information and this type of application prohibited any credible statement about central calls, input/output, or D_i as an error predictor. The study should be repeated on an entire system, which has an even distribution of the elements of D_i. To ensure data integrity, the developers should be at hand, and any assumptions about the system should be cleared through them. In analyzing a stable system, where the researcher can control the external variables, a more accurate study can be performed. If this type of system can be obtained, verification of the D_i metric and components as an error-prone module locator can be completed successfully.

Bibliography

Baker, Albert L. and Zweben, Stuart H. A Comparison of Control Flow Complexity. IEEE Transactions on Software Engineering. Vol SE-6. No. 6. November 1980.

Barnes, J.G.P. Programming in Ada. Third Edition. Addison-Wesley Publishing. Menlo Park, California. 1989.

Card, David, Church, Victor, and Agresti, William. An Empirical Study of Software Design Practices. IEEE Transactions on Software Engineering. Vol SE-12. No. 2. February 1986.

Chol, Song C. and Scacchi, Walt. Extracting and Restructuring the Design of Large Systems. IEEE Software. January 1990.

Conte, S.D., Dunsmore H.E., and Shen V.Y.. Software Engineering Metrics and Models. The Benjamin/Cummings Publishing Company. Menlo Park, California. 1986.

Dunsmore, H.E. Evidence Supports Some Truisms, Belies Others. IEEE Software.

Glass, Robert. Persistent Software Errors. IEEE Transactions on Software Engineering. Vol. SE-7. No 2. March 1981.

Holub, Allen. Compiler Design in C. Prentice-Hall Publishing. 1990. Englewood Cliffs, New Jersey.

LI, H.F. and Chueng, W.K. An Empirical Study of Software Metrics. IEEE Transactions on Software Engineering. Vol. SE-13. No. 6. June 1987.

Norusis, Marija. SPSS/PC+ Studentware. SPSS Publishing 1988. Chicago, Illinois.

Rombach, H. Dieter. Design Measurement: Some Lessons Learned. IEEE Software. March 1990.

Withrow, Carol. Error Density and Size in Ada Software. IEEE Software. January 1990.

Zage, Wayne M. and Zage, Dolores M. Relating Design Metrics to Software Quality: Some Empirical Results. SERC Technical Report TR-74-P, May 1990.

Zage, Wayne M. Design Metrics Status Report. May 1990.

Appendix A

Thesis Program Code Listing

This appendix contains the code for the Thesis program. The file was generated by the Microsoft C 6.0 compiler.

```
Line#  Source Line

     1
     2 /*************************/
     3 /*                       */
     4 /*  THESIS.C               */
     5 /*                       */
     6 /*  Code file for thesis  */
     7 /*  program.               */
     8 /*                       */
     9 /*************************/
    10
    11 #include <stdio.h>
    12 #include <malloc.h>
    13 #include <memory.h>
    14
    15 /* Utility Defines */
    16 #define FILENAME "EMPLOYEE.DBS"
    17
    18 /* Global Typredefs*/
    19 typedef struct employee_type {
    20                                 char name[20];
    21                                 char address[30];
    22                                 int age, hight, weight;
    23                                 struct employee_type *next;
    24                               } employee_record;
    25
    26
    27 /* Functional Prototypes */
    28 employee_record *load_file(void);
    29 void print_records( employee_record *x_ptr);
    30 void free_records( employee_record *x_ptr);
    31
    32
    33 /* Data Declarations */
    34 employee_record *list_ptr;
    35
    36
    37 /* Functions */
    38 int main(void)
    39   {
```

```
40      printf("Employee DataBase Program Starting ...\n");
41      list_ptr = load_file();
42
43      if (list_ptr != NULL)
44        {
45          print_records(list_ptr);
46          printf("Employee Database Successfully Read.\n");
47        }
48      else
49        {
50          printf("Error No Database found.\n");
51        }
52      printf("Employee DataBase Program Terminated.\n");
53      return ( 1);
54    }
55
56  employee_record *load_file(void)
57    {
58      FILE *fp;
59      employee_record *top, *t_ptr, temp;
60      int counter;
61
62      top = NULL;
63      if ((fp = fopen(FILENAME, "r+")) != NULL)
64        {
65          counter = 5;
66          while (counter == 5)
67            {
68              if ((counter = fscanf( fp, "%20.20s%30.30s %d %d %d", &temp
    .name, &temp.address,
69                                      &temp.age, &temp.hight, &temp.weight
    )) == 5)
70                {
71                  if ((t_ptr=malloc(sizeof(employee_record))) == NULL)
72                    {
73                      printf("Error Allocating Memeory - Out of Memory\n
    ");
74                      counter = -1;
75                    }
76                  else
77                    {
78                      memcpy( t_ptr, &temp, sizeof(employee_record));
79                      if (top == NULL)
80                        {
81                          top = t_ptr;
82                          t_ptr -> next = NULL;
83                        }
84                      else
85                        {
86                          t_ptr -> next = top;
87                          top = t_ptr;
88                        }
89                    }
90                }
91            }
```

```
92        }
93      else
94        {
95          printf("Error opening database file.\n");
96        }
97      return ( top);
98    }
```

load_file Local Symbols

Name	Class	Type	Size	Offset	Register
t_ptr	auto			-004a	
fp.	auto			-0046	
temp.	auto			-0042	
top	auto			-0006	
counter	auto			-0002	

```
99
100 void print_records(employee_record *x_ptr)
101   {
102     if (x_ptr == NULL)
103       printf("No Records Found or Unable to find Database.\n");
104     else
105       while ( x_ptr != NULL)
106         {
107           printf("\n");
108           printf(" Name    : %-20.20s\n", x_ptr -> name);
109           printf(" Address : %-20.20s\n", x_ptr -> address);
110           printf(" Age     : %-3d\n", x_ptr -> age);
111           printf(" Hight   : %-3d\n", x_ptr -> hight);
112           printf(" Weight  : %-3d\n", x_ptr -> weight);
113           printf("\n");
114           x_ptr = x_ptr -> next;
115         }
116   }
```

print_records Local Symbols

Name	Class	Type	Size	Offset	Register
x_ptr	param			0006	

```
117
118 void free_records( employee_record *x_ptr)
119   {
120     employee_record *temp_ptr;
121
122     while (x_ptr != NULL)
123       {
124         temp_ptr = x_ptr;
125         x_ptr = x_ptr -> next;
```

```
126             free( temp_ptr);
127         }
128     }
```

free_records Local Symbols

Name	Class	Type	Size	Offset	Register
temp_ptr.	auto			-0004	
x_ptr	param			0006	

Global Symbols

Name	Class	Type	Size	Offset
fopen	extern	far function	***	***
free.	extern	far function	***	***
free_records.	global	far function	***	0226

Global Symbols

Name	Class	Type	Size	Offset
fscanf.	extern	far function	***	***
list_ptr.	common	far pointer	4	***
load_file	global	far function	***	005e
main.	global	far function	***	0000
malloc.	extern	far function	***	***
memcpy.	extern	far function	***	***
print_records	global	far function	***	0166
printf.	extern	far function	***	***

Code size = 0262 (610)
Data size = 0190 (400)
Bss size = 0000 (0)

No errors detected

Appendix B

Thesis Program Structure Chart and Flow Graphs

Figure B-1 contains the structure chart for the thesis program. Figures B-2, B-3, B-4, B-5 contain flow graphs generated from modules *main*, *load_records*, *print_records*, and *free_records*.

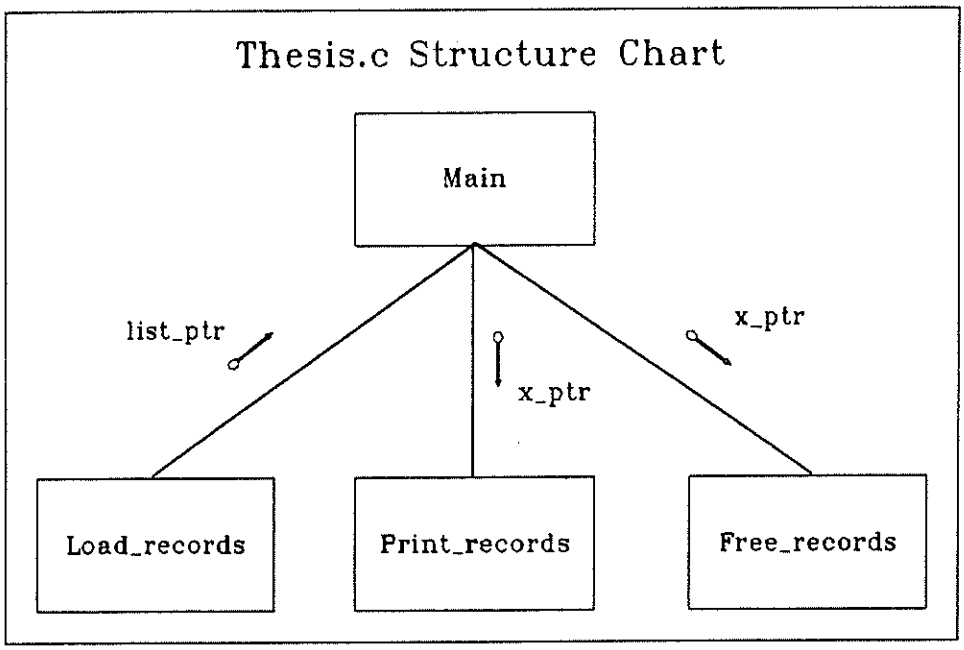

Figure B-1

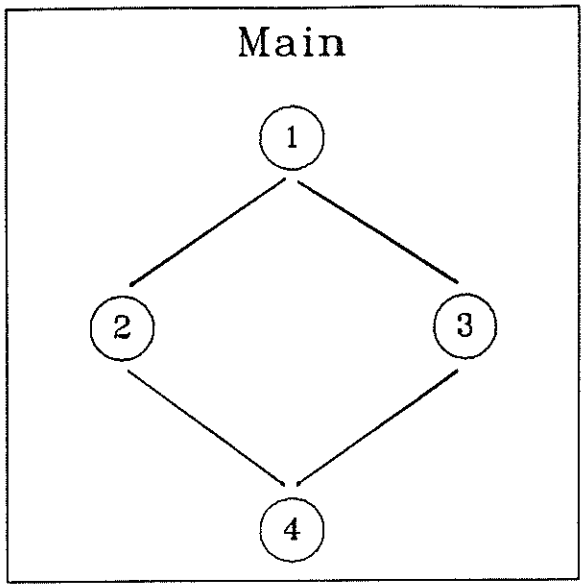

Figure B-2

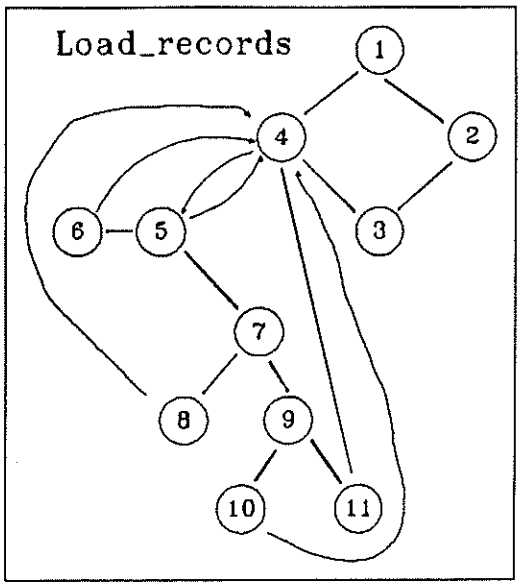

Figure B-3

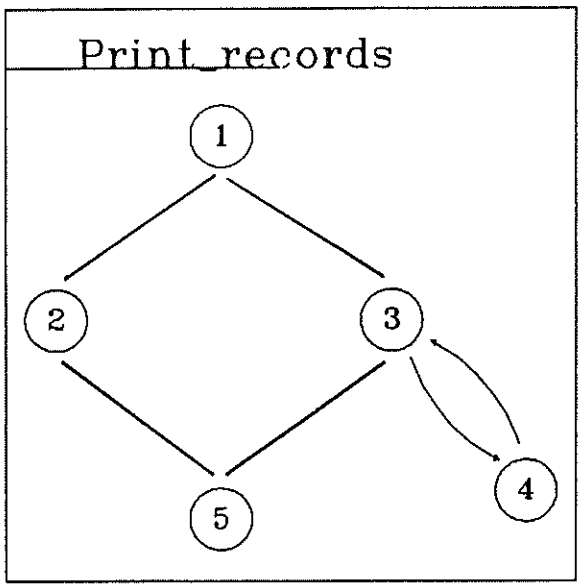

Figure B-4

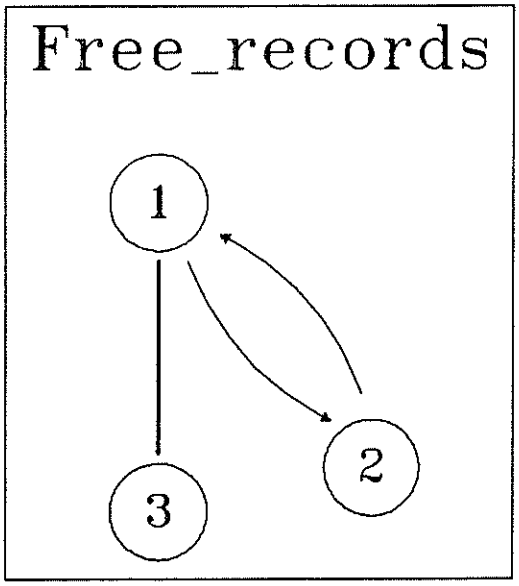

Figure B-5

In table B-1, listed below, are the line numbers corresponding to the nodes of the flow graphs.

Flow Graph Line Numbersa

Module Main

Node	Line Numbers
1	39 - 43
2	44 - 48
3	49 - 51
4	52 - 54

Module Load_records

Node	Line Numbers
1	62 - 63
2	95
3	97
4	65 - 66
5	68 - 69
6	70
7	71
8	73-74
9	78 - 79
10	82 - 83
11	87 - 88

Module Print_records

Node	Line Number
1	102
2	103
3	105
4	107 - 114
5	116

Module Free_records

Node	Line Number
1	122
2	124 - 126
3	128

Table B-1

Appendix C - Pilot Study Project Specifications.

CS680 Project 1

Requirements :

For a certain company, the method by which the pay for each employee is computed depends on whether that employee is classified as an Office employee, a Factory employee, or a Salesperson. Suppose the a file of employee records is maintained is which each record is variant depending on the classification. The following is the format of these data.

Name (20 Characters)
Social Security Number (integer)
Age (Integer)
Number of Dependents (integer)
Employee Code (O, F, or S)
Hourly Rate, if employee is Factory worker
Annual Salary, if employee is office worker
A Base pay and commission if employee is salesperson

The program should be menu driven allowing at least the following options, which are selected by the user.

Insert the record for a new employee
Update the record of a existing employee
(e.g. allow change of name, SS#, etc.)
Retrieve and display the record for a specified employee (for Name or SS#)
List the records (or perhaps selected items in the records) in order. This option should allow these sub-options
-- list all employees
-- list only office employees
-- list only factory employees
-- list only salespeople
To copy records into a permanent file
To delete the record of an employee from the file

Produce a data flow diagram, the structure chart, and running code in any language. Your program should be as user-friendly as possible.

Appendix D - Pilot Study Project Characteristics

MP-1 System Statistics

Modules Names	Errors	SLOC	CC	DSM	I/O	D.
Interface	0	14	0	0	11	11
Print-Menu	0	19	2	9	3	14
GetRecord	0	37	0	0	20	20
Ordersearch	0	28	0	3	2	5
Copy-list-to-file	0	13	1	3	2	6
Insert †	16	35	3	11	4	18
Update * †	10	99	2	2	45	49
Delete	1	22	3	5	4	12
Write-data	0	22	0	0	12	12
Search	1	16	2	3	3	8
Print_menu2	0	13	0	0	10	10
List-office	0	15	1	5	3	9
List_factory	0	15	1	5	3	9
List-sales	0	15	1	5	3	9
List-all	0	14	1	5	3	9
List	0	21	4	0	4	8
Main	0	35	6	0	10	16

* Highlighted by D unit weighting scheme

† Highlighted by alternate D weighting scheme

Table D-1

Table D-2

MP-2 System Statistics

Module Name	ERRORS	SLOC	CC	DSM	I/O	D_s
Find_element	0	15	0	10	0	10
Insert	0	13	0	12	0	12
Create_order*	0	49	6	26	18	50
Print_menu	0	13	0	0	11	11
Add_element*†	6	73	3	33	25	61
Find_employee	0	15	0	8	4	12
Update_emp*†	6	121	1	25	41	67
Update	0	38	2	2	9	13
Delete	0	17	1	11	0	12
Delete_element	0	44	1	2	14	17
Emp_copy*	0	47	3	15	23	41
Write_list	0	15	1	10	6	17
List_all	0	24	1	10	10	21
List_office	0	21	1	7	9	17
List_sales	0	21	1	7	9	17
List_factory	0	21	1	7	9	17
List	0	25	5	0	11	16
List_copy	0	20	3	0	9	12
Final_instruct	0	9	1	0	7	8
Main_menu	0	19	5	0	5	10
Main	0	33	9	5	0	14

* Highlighted by D unit weighting scheme

† Highlighted by alternate D weighting scheme

Table D-3

MP-3 System Statistics

Module Name	ERRORS	SLOC	CC	DSM	I/O	D
Main	0	10	9	0	0	9
Confirm	0	23	12	0	5	17
Msg_tem	0	17	12	0	7	19
Show_exsten	0	11	4	0	4	8
Screen_init	0	15	9	0	8	17
Init_cmds	0	11	9	9	0	18
Operations	0	21	8	0	0	8
Main_menu	0	15	8	6	2	16
Display_select	0	5	1	1	1	3
Select_cmd	0	20	4	0	4	8
Insert_record*+	5	44	34	16	28	78
Update_record	0	27	12	11	2	25
Update_command	0	18	4	0	4	8
B_update_list	0	7	0	5	0	5
Update_show_high	0	21	5	16	1	21
Update_attrib*+	6	72	54	35	38	127
Retrieve_record	0	18	8	3	0	11
Specify_record	0	21	13	3	5	21
Find_record	1	12	1	7	0	8
Display_record	1	19	15	8	6	29
Deltete_record	0	31	12	14	4	30
Build_del_list	0	7	0	5	0	5
Del_show_high	0	17	4	8	1	13
Del_command	0	19	7	4	4	15
Save_file	0	10	4	6	3	13
Read_file	0	20	9	8	3	20
List_commands	0	30	12	2	2	16
Ldisplay_sel	0	27	1	3	1	5
Perf_list	0	16	3	0	0	3
Build_list	0	7	1	7	0	8
Show_list	0	17	8	8	3	19
Show_high	0	17	6	10	1	17
Lselect_command	0	7	4	0	4	8

* Highlighted by D unit weighting scheme

+ Highlighted by alternate D weighting scheme

Appendix E - STANFINS D_i Metric Counts

Easy Programs

EASY 1

CC	DSM	I/O	ERRS	Module Name
9	13	0	0	Avzhgpe5_Aspect
1	86	0	1	Avzhgpe5_Aspect.Set_Initial_Screen
0	23	0	0	Ehgp_View.Appl_Fnd_Cmt_Doc.Make_Prime_Kev
0	1	0	0	Ehgp_View.Appl_Fnd_Cmt_Doc.Initialized
2	61	0	0	Ehgp_View.Retrieve
2	0	0	0	Ehgp_View.Operation_Status
0	21	0	2	Avzhgpe5_Aspect.Process_Steps
1	79	0	3	Avzhgpe5_Aspect.Process_Steps.Retrieve_Record_And_Update
0	35	0	0	Avzhgpe5_Aspect.Process_Steps.Retrieve_Record_And_Update.Lo
1	30	0	0	Ehgp_View.Modify_Record
0	13	0	0	Avzhgpe5_Aspect.Ehgp_Continue_From_A
0	11	0	0	Interactive_Input_Comm_Area.Transfer_Control_And_Exit
0	1	0	0	Avzhgpe5_Aspect.No_Local_Process

EASY 2

CC	DSM	I/O	ERRS	Module Name
7	12	0	0	Avzubpe4_Aspect
0	13	0	0	Avzubpe4_Aspect.Set_Initial_Screen
6	21	0	0	Avzubpe4_Aspect.Process_Steps.Prepare_For_Step_2
1	117	0	0	Avzubpe4_Aspect.Process_Steps.Edit_Kev_Data.Build_Fnd_M_jo_Inf
0	3	0	0	Eubp_View.Appl_Fnd_Mjo.Initialized
0	15	0	0	Eubp_View.Appl_Fnd_M_jo.Make_Prime_Kev
3	64	0	0	Eubp_View.Retrieve
2	0	0	0	Eubp_View.Operation_Status
1	62	0	0	Avzubpe4_Aspect.Process_Steps.Edit_Kev_Data.Build_M_jo_Dst_Inf
0	5	0	0	Eubp_View.Appl_Mjo_Dst.Initialized
1	57	0	1	Avzubpe4_Aspect.Process_Steps.Edit_Kev_Data.Build_All_Id_Inf
0	11	0	0	Eubp_View.Appl_All_Id.Make_Prime_Kev
0	1	0	0	Eubp_View.Appl_All_Id.Initialized
0	106	0	3	Avzubpe4_Aspect.Process_Steps.Validate_Screen_Amounts
2	7	0	0	Avzubpe4_Aspect.Process_Steps.Prepare_For_Step_3
0	47	0	0	Avzubpe4_Aspect.Process_Steps.Process_Change_Amount
0	48	0	0	Avzubpe4_Aspect.Process_Steps.Process_Confirmation.Validate_Pa
1	92	0	0	Avzubpe4_Aspect.Process_Steps.Process_Confirmation.Modifv_Fnd
2	11	0	0	Eubp_View.Release_Record
1	39	0	0	Eubp_View.Modifv_Record
2	57	0	0	Avzubpe4_Aspect.Process_Steps.Process_Conf_irmation.Add_Fnd
0	3	0	0	Eubp_View.Appl_Fnd_Dst_Pb.Initialized
1	28	0	0	Eubp_View.Add_Record
0	80	0	0	Avzubpe4_Aspect.Process_Steps.Process_Confirmation.Build_Scrn
0	1	0	0	Avzubpe4_Aspect.No_Local_Process

EASY 3

CC	DSM	I/O	ERRS	Module Name
6	3	0	0	Avzcrpm5_Aspect
1	13	0	2	Avzcrpm5_Aspect.Put_Report_Header
0	6	0	0	Avzcrpm5_Aspect.Convert_To_Sjo
0	105	0	20	Avzcrpm5_Aspect.Edit_Data
1	98	0	27	Avzcrpm5_Aspect.Edit_Data.Determine_Atg_Month_Info
0	12	0	0	Mcrp_View.Appl_Atg_Month.Make_End_Dt_Kev
0	3	0	0	Mcrp_View.Appl_Atg_Month.Initialized

3	76	0	0	Merp_View.Retrieve
0	2	0	0	Merp_View.Operation_Status
2	56	0	6	Avzcrpm5_Aspect.Edit_Data.Determine_Input_Errors
2	142	0	5	Avzcrpm5_Aspect.Edit_Data.Validate_Input_Fields
0	1	0	0	Merp_View.Appl_Eor_Tbl.Initialized
0	11	0	0	Merp_View.Appl_Eor_Tbl.Make_Prime_Key
3	63	0	0	Merp_View.Read_Index_Only
6	200	0	12	Avzcrpm5_Aspect.Edit_Data.Verify_Database_Records
0	15	0	0	Merp_View.Appl_Fnd_S_jo.Make_Prime_Key
0	1	0	0	Merp_View.Appl_Fnd_Sjo.Initialized
0	15	0	0	Merp_View.Appl_Rmb_Ord.Make_Prime_Key
0	3	0	0	Merp_View.Appl_Rmb_Ord.Initialized
0	1	0	0	Merp_View.Appl_All_Id.Initialized
0	7	0	0	Merp_View.Appl_Fnd_Mjo.Initialized
0	27	0	0	Merp_View.Appl_Exec_Cmt_Obg.Make_Prime_Key
0	11	0	0	Merp_View.Appl_Exec_Cmt_Obg.Initialized
0	11	0	0	Merp_View.Appl_All_Id_Make_Prime_Key
5	218	0	27	Avzcrpm5_Aspect.Update_Files
0	3	0	0	Merp_View.Appl_Lmt_Mjo.Initialized
0	5	0	0	Merp_View.Appl_Exec_Rpt.Initialized
2	72	0	2	Avzcrpm5_Aspect.Update_Files.Build_Batch_Record
1	35	0	6	Avzcrpm5_Aspect.Update_Files.Print_Update_Error
1	64	0	4	Avzcrpm5_Aspect.Update_Files.Build_Lmt_Mjo_Buffer
0	1	0	0	Merp_View.Appl_Limit_Code_Tbl.Initialized
0	15	0	0	Merp_View.Appl_Limit_Code_Tbl.Make_Prime_Key
0	19	0	0	Merp_View.Appl_Lmt_Mjo.Make_Prime_Key
0	67	0	0	Avzcrpm5_Aspect.Update_Files.Build_Rmb_Ord_Buffer
0	73	0	6	Avzcrpm5_Aspect.Update_Files.Build_Exec_Cmt_Obg_Buffer
0	1	0	0	Merp_View.Appl_Cus_Tvp_Tbl.Initialized
0	1	0	0	Merp_View.Appl_Exec_Grp_Asg.Initialized
0	7	0	0	Merp_View.Appl_Cus_Tvp_Tbl.Make_Prime_Key
0	19	0	0	Merp_View.Appl_Exec_Grp_Asg.Make_Prime_Key
1	111	0	13	Avzcrpm5_Aspect.Update_Files.Build_Fnd_Mjo_Buffer
2	29	0	0	Merp_View.Release_Record
0	1	0	0	Merp_View.Record_Is_Locked
0	100	0	3	Avzcrpm5_Aspect.Update_Files.Build_Exec_Rpt_Buffer
0	65	0	0	Merp_View.Appl_Exec_Rpt.Make_Prime_Key
1	54	0	2	Avzcrpm5_Aspect.Update_Files.Modify_All_Lgr_Buffer
0	16	0	0	Merp_View.Appl_All_Lgr.Make_Prime_Key
0	3	0	0	Merp_View.Appl_All_Lgr.Initialized
0	11	0	0	Merp_View.Appl_Exec_Pst.Make_Prime_Key
0	3	0	0	Merp_View.Appl_Exec_Pst.Initialized
0	9	0	0	Merp_View.Appl_Exec_Eor_Tran.Initialized
1	40	0	0	Merp_View.Add_Record
1	40	0	0	Merp_View.Modify_Record
0	24	0	0	Avzcrpm5_Aspect.Update_Files.Fix_Ode
0	18	0	0	Avzcrpm5_Aspect.Close_Edit_Report

EASY 4

CC	DSM	I/O	ERRS	Module Name
7	12	0	0	Avzw1pf3_Aspect
1	116	0	1	Avzw1pf3_Aspect.Display_Initial_Screen
0	1	0	0	Fw1p_View.Appl_Fxd_Ast.Initialized
0	11	0	0	Fw_1_p_View.Appl_Fxd_Ast.Make_Prime_Key
0	6	0	1	Avzw1pf3_Aspect.Display_Initial_Screen.Blank_In_Record
0	21	0	2	Avzw1pf3_Aspect.Display_Initial_Screen.Set_Screen
3	61	0	0	Fw_1_p_View.Retrieve
2	0	0	0	Fw_1_p_View.Operation_Status
0	0	0	1	Avzw_1_pf_3_Aspect.No_Local_Process

EASY 5

CC	DSM	I/O	ERRS	Module Name
4	5	0	0	Avzk_1_pg7_Aspect
0	69	0	0	Avzk1pg7_Aspect.Do_Top_Edits
0	1	0	0	Gk1p_View.Appl_Dssn_Un_Tbl.Initialized

0	7	0	0	Gk_1_p_View.Appl_Dssn_Un_Tbl.Make_Prime_Key
3	50	0	0	Gk1p_View.Read_Index_Only
2	0	0	0	Gk_1_p_View.Operation_Status
0	17	0	0	Avzk_1_pg7_Aspect.Finish_Building_Record
1	21	0	0	Avzk_1_pg7_Aspect.Lock_All_Top_Fields

EASY 6

CC	DSM	I/O	ERRS	Module Name
4	5	0	0	Avzlspg9_Aspect
0	300	0	9	Avzlspg9_Aspect.Do_Top_Edits
0	44	0	2	Avzlspg9_Aspect.Do_Top_Edits.Valid_Input
0	160	0	0	Avzlspg9_Aspect.Finish_Building_Record
0	60	0	2	Avzlspg9_Aspect.Lock_All_Top_Fields

Easy 7

CC	DSM	I/O	ERRS	Module Name
9	13	0	0	Avzhlpe5_Aspect
0	35	0	0	Avzhlpe5_Aspect.Transfer_Logic
0	45	0	0	Interactive_Input_Comm_Area.Transfer_Control_And_Exit
0	17	0	1	Avzhlpe5_Aspect.Set_Key_Fields
0	30	0	0	Avzhlpe5_Aspect.Set_Input_Fields
0	35	0	0	Avzhlpe5_Aspect.Set_Bottom_Fields
0	28	0	0	EhlmO.Set_Initial_Screen
0	23	0	0	Ehlp_View.Appl_Fnd_Cmt_Req.Make_Prime_Key
3	69	0	0	Ehlp_View.Retrieve
0	40	0	0	Avzhlpe5_Aspect.Setup_Cmt_Req_Add
0	80	0	0	Avzhlpe5_Aspect.Setup_Cmt_Req_Modify
1	41	0	0	Avzhlpe5_Aspect.Send_Next_Screen
0	1	0	0	Ehlp_View.Appl_Fnd_Cmt_Req.Initialized
2	0	0	0	Ehlp_View.Operation_Status
1	39	0	0	Avzhlpe5_Aspect.Send_Previous_Screen
0	35	0	0	Avzhlpe5_Aspect.Process_Steps
0	53	0	0	Avzhlpe5_Aspect.Process_Steps.Edit_Inputs
0	36	0	1	Avzhlpe5_Aspect.Process_Steps.Edit_Inputs.Edit_Required_Fi
1	50	0	0	Avzhlpe5_Aspect.Process_Steps.Edit_Inputs.Get_Fnd_Sjo
0	1	0	0	Ehlp_View.Appl_Fnd_Sjo.Initialized
0	15	0	0	Ehlp_View.Appl_Fnd_Sjo.Make_Prime_Key
1	79	0	0	Avzhlpe5_Aspect.Process_Steps.Edit_Inputs.Get_Fnd_Cmt_Doc
0	17	0	0	Avzhlpe5_Aspect.Process_Steps.Update_Cmt_Document.Get_All
1	47	0	0	Avzhlpe5_Aspect.Process_Steps.Update_Cmt_Document.Validate
0	3	0	0	Ehlp_View.Appl_Ccc_Jo_Corr_Tbl.Initialized
0	11	0	0	Ehlp_View.Appl_Ccc_Jo_Corr_Tbl.Make_Prime_Key
0	24	0	0	Avzhlpe5_Aspect.Process_Steps.Update_Cmt_Document.Validate
0	29	0	0	Avzhlpe5_Aspect.Process_Steps.Update_Cmt_Document.Update_F
1	38	0	0	Ehlp_View.Modify_Record
2	122	0	2	Avzhlpe5_Aspect.Process_Steps.Update_Cmt_Document.Build_Re
0	5	0	0	Ehlp_View.Appl_Fnd_Mjo.Initialized
0	15	0	0	Ehlp_View.Appl_Fnd_M_jo.Make_Prime_Key
0	1	0	0	Ehlp_View.Appl_Rmb_Ord.Initialized
0	11	0	0	Ehlp_View.Appl_Rmb_Ord.Make_Prime_Key
1	62	0	0	Avzhlpe5_Aspect.Process_Steps.Update_Cmt_Document.Get_Req
4	122	0	0	Avzhlpe5_Aspect.Process_Steps.Update_Req_Document
0	3	0	0	Ehlp_View.Appl_Ga_Sys_Isl.Initialized
2	54	0	2	Avzhlpe5_Aspect.Process_Steps.Update_Req_Document.Edit_Req
0	98	0	6	Avzhlpe5_Aspect.Process_Steps.Update_Req_Document.Validate
0	1	0	0	Ehlp_View.Appl_Eor_Tbl.Initialized
0	11	0	0	Ehlp_View.Appl_Eor_Tbl.Make_Prime_Key
0	55	0	0	Avzhlpe5_Aspect.Process_Steps.Update_Req_Document.Retrieve
0	98	0	0	Avzhlpe5_Aspect.Process_Steps.Update_Req_Document.Retrieve
0	11	0	0	Ehlp_View.Appl_Ga_Svs_Isl.Make_Prime_Key
0	35	0	0	Avzhlpe5_Aspect.Process_Steps.Clear_Bottom_Fields
0	3	0	0	Avzhlpe5_Aspect.Quit
0	14	0	0	Avzhlpe5_Aspect.Continue_Processing
0	0	0	0	Avzhlpe5_Aspect.No_Local_Process

Medium Programs

Medium 1

CC	DSM	I/O	ERRS	Module Name
4	5	0	0	Avzxmpg8_Aspect
0	19	0	6	Avzxmpg8_Aspect.Pss_Parm.Put
4	77	0	27	Avzxmpg8_Aspect.Do_Edits
0	10	0	6	Avzxmpg8_Aspect.Do_Edits.Set_Required_Field_Attr
1	22	0	0	Avzxmpg8_Aspect.Do_Edits.Validate
0	3	0	0	Gxmp_View.Appl_Atg_Month.Initialized
0	15	0	0	Gxmp_View.Appl_Atg_Month.Make_Prime_Key
3	63	0	0	Gxmp_View.Retrieve
2	0	0	0	Gxmp_View.Operation_Status
0	0	0	0	Avzxmpg8_Aspect.Do_Edits.Invalid_Year_End_Idc
0	45	0	19	Avzxmpg8_Aspect.Do_Edits.Set_Invalid_Field_Attr
1	22	0	0	Avzxmpg8_Aspect.Do_Edits.Batch_Open
0	1	0	0	Gxmp_View.Appl_Bat_Inp_Hdr.Initialized
0	15	0	0	Gxmp_View.Appl_Bat_Inp_Hdr.Make_Prime_Key
0	9	0	2	Avzxmpg8_Aspect.Finish_Building_Record
1	10	0	2	Avzxmpg8_Aspect.Lock_Fields

Medium 2

CC	DSM	I/O	ERRS	Module Name
4	5	0	0	Avzr2pg2_Aspect
0	91	0	3	Avzr2pg2_Aspect.Pss_Parm.Put
1	103	0	26	Avzr2pg2_Aspect.Do_Top_Edits
0	51	0	18	Avzr2pg2_Aspect.Do_Top_Edits.Set_Required_Field_Attr
1	10	0	6	Avzr2pg2_Aspect.Do_Top_Edits.Invalid_Prf_Cd
0	1	0	0	Gr2p_View.Appl_Std_Inx.Initialized
0	11	0	0	Gr2p_View.Appl_Std_Inx.Make_Prime_Key
3	57	0	0	Gr2p_View.Read_Index_Only
2	0	0	0	Gr2p_View.Operation_Status
0	5	0	0	Avzr2pg2_Aspect.Do_Top_Edits.Invalid_Rpt_Idc
0	2	0	0	Avzr2pg2_Aspect.Do_Top_Edits.Invalid_Rpt_Pat_Idc
1	16	0	0	Avzr2pg2_Aspect.Do_Top_Edits.Invalid_Atg_Dt
0	1	0	0	Gr2p_View.Appl_Atg_Month.Initialized
0	12	0	0	Gr2p_View.Appl_Atg_Month.Make_End_Dt_Key
0	11	0	2	Avzr2pg2_Aspect.Do_Top_Edits.Required_Rpt_Pat_Fields
0	11	0	0	Avzr2pg2_Aspect.Do_Top_Edits.Invalid_Rpt_Pat_Fields
1	24	0	0	Avzr2pg2_Aspect.Do_Top_Edits.Invalid_Bs
0	1	0	0	Gr2p_View.Appl_Bs_Edit_Tbl.Initialized
0	15	0	0	Gr2p_View.Appl_Bs_Edit_Tbl.Make_Prime_Key
1	23	0	1	Avzr2pg2_Aspect.Do_Top_Edits.Invalid_Ams
0	1	0	0	Gr2p_View.Appl_Ams_Tbl.Initialized
0	7	0	0	Gr2p_View.Appl_Ams_Tbl.Make_Prime_Key
1	24	0	1	Avzr2pg2_Aspect.Do_Top_Edits.Invalid_Ams_Sp_Prj
0	1	0	0	Gr2p_View.Appl_Ams_Sp_Prj_Tbl.Initialized
0	11	0	0	Gr2p_View.Appl_Ams_Sp_Prj_Tbl.Make_Prime_Key
1	24	0	1	Avzr2pg2_Aspect.Do_Top_Edits.Invalid_Sys_Cpe
0	1	0	0	Gr2p_View.Appl_Ams_Sys_Ttl_Tbl.Initialized
0	7	0	0	Gr2p_View.Appl_Ams_Sys_Ttl_Tbl.Make_Prime_Key
1	23	0	4	Avzr2pg2_Aspect.Do_Top_Edits.Invalid_Uic
0	1	0	0	Gr2p_View.Appl_Uic_Mst_Tbl.Initialized
0	11	0	0	Gr2p_View.Appl_Uic_Mst_Tbl.Make_Prime_Key
2	86	0	0	Avzr2pg2_Aspect.Do_Top_Edits.Check_All_Id_Wss_Id
0	1	0	0	Gr2p_View.Appl_All_Id.Initialized
0	23	0	0	Gr2p_View.Appl_All_Id.Make_Cas_All_Key
0	1	0	0	Gr2p_View.Appl_Fnd_Sjo.Initialized
3	70	0	0	Gr2p_View.Retrieve
0	23	0	0	Gr2p_View.Appl_Fnd_Sjo.Make_Wss_Seq_Key
0	160	0	10	Avzr2pg2_Aspect.Do_Top_Edits.Set_Invalid_Field_Attr
1	34	0	9	Avzr2pg2_Aspect.Finish_Building_Record
1	40	0	0	Avzr2pg2_Aspect.Lock_All_Top_Fields

Medium 3

CC	DSM	I/O	ERRS	Module Name
8	11	0	0	AvzO6pe7_Aspect
0	35	0	0	AvzO6pe7_Aspect.Set_Key_Fields
0	25	0	0	AvzO6pe7_Aspect.Set_Data_Line
0	53	0	0	AvzO6pe7_Aspect.Clear_Screen
1	42	0	4	AvzO6pe7_Aspect.Record_Valid
0	1	0	0	E06p_View.Appl_Misc_Sjo.Initialized
0	19	0	0	E06p_View.Appl_Misc_Sjo.Make_Prime_Key
3	56	0	0	E06p_View.Read_Index_Only
2	0	0	0	E06p_View.Operation_Status
1	204	0	1	AvzO6pe7_Aspect.Fill_Screen
0	7	0	0	E06p_View.Appl_Dssn_Doc_Oth.Initialized
0	35	0	0	E06p_View.Appl_Dssn_Doc_Oth.Make_Prime_Key
3	69	0	0	E06p_View.Retrieve
1	199	0	0	AvzO6pe7_Aspect.Fill_Screen_Reversed
0	32	0	0	AvzO6pe7_Aspect.Set_Initial_Screen
0	23	0	0	AvzO6pe7_Aspect.Send_Previous_Screen
0	23	0	0	AvzO6pe7_Aspect.Send_Next_Screen
0	28	0	0	AvzO6pe7_Aspect.Process_Steps
0	39	0	0	AvzO6pe7_Aspect.Process_Steps.Edit
0	62	0	0	AvzO6pe7_Aspect.Process_Steps.Edit.Edit_Required_Fields
1	85	0	0	AvzO6pe7_Aspect.Process_Steps.Edit.Check_Dssn_Doc_0th
0	105	0	0	AvzO6pe7_Aspect.Process_Steps.Update
0	15	0	0	AvzO6pe7_Aspect.Process_Steps.Update.Block_Entered
0	44	0	0	AvzO6pe7_Aspect.Process_Steps.Update.Edit_Required_Fields
0	28	0	0	AvzO6pe7_Aspect.Process_Steps.Update.Validate_Rej_Tfo_Cd
0	1	0	0	E06p_View.Appl_Dssn_Tbl.Initialized
0	7	0	0	E06p_View.Appl_Dssn_Tbl.Make_Prime_Key
0	1	0	0	E06p_View.Appl_Dssn_Un_Tbl.Initialized
0	7	0	0	E06p_View.Appl_Dssn_Un_Tbl.Make_Prime_Key
3	124	0	0	AvzO6pe7_Aspect.Process_Steps.Update.Validate_Sjo
0	1	0	0	E06p_View.Appl_Fnd_Sjo.Initialized
0	15	0	0	E06p_View.Appl_Fnd_Sjo.Make_Prime_Key
1	121	0	1	AvzO6pe7_Aspect.Process_Steps.Update.Modify_Dssn_Doc_0th
1	200	0	12	AvzO6pe7_Aspect.Process_Steps.Update.Modify_Dssn_Doc_Oth.A
0	5	0	0	E06p_View.Appl_Bat_Inp_Dtl.Initialized
1	33	0	0	E06p_View.Add_Record
2	16	0	0	E06p_View.Release_Record
1	33	0	0	E06p_View.Modify_Record
0	24	0	0	AvzO6pe7_Aspect.Process_Steps.Refresh_Screen
0	0	0	0	AvzO6pe7_Aspect.No_Local_Process
0	29	0	4	AvzO6pe7_Aspect.Quit
1	35	0	0	AvzO6pe7_Aspect.Quit.Create_Mbjp_Dtl_Hdr
1	30	0	0	AvzO6pe7_Aspect.Quit.Create_Mbjp_Dtl_Trl
1	32	0	0	AvzO6pe7_Aspect.Quit.Create_Bat_Inp_Hdr
0	9	0	0	E06p_View.Appl_Irr_Queue.Initialized
0	3	0	0	E06p_View.Appl_Irr_Rpt_Default.Initialized
0	15	0	0	E06p_View.Appl_Irr_Rpt_Default.Make_Prime_Key

Medium 4

CC	DSM	I/O	ERRS	Module Name
7	12	0	0	Avzgope5_Aspect
0	13	0	1	Avzgope5_Aspect.Set_Initial_Screen
0	100	0	2	Avzgope5_Aspect.Process_Steps
2	85	0	3	Avzgope5_Aspect.Process_Steps.Retrieve_Fnd_S_jo_Rec
0	15	0	0	Egop_View.Appl_Fnd_Sjo.Make_Prime_Key
0	1	0	0	Egop_View.Appl_Fnd_Sjo.Initialized
0	78	0	0	Egop_View.Retrieve3
2	0	0	0	Egop_View.Operation_Status
1	21	0	6	Avzgope5_Aspect.Process_Steps.Get_Fnd_M_jo_Rec
0	15	0	0	Egop_View.Appl_Fnd_Mjo.Make_Prime_Key
0	9	0	0	Egop_View.Appl_Fnd_Mjo.Initialized
0	32	0	0	Avzgope5_Aspect.Process_Steps.Set_Top_Attr

0	20	0	0	Avzgope5_Aspect.Process_Steps.Set_Bottom_Attr
0	56	0	6	Avzgope5_Aspect.Process_Steps.Edit_Top_Fields
0	66	0	0	Avzgope5_Aspect.Process_Steps.Edit_Top_Fields.Required_Fie
1	52	0	5	Avzgope5_Aspect.Process_Steps.Edit_Top_Fields.Move_Allotme
0	11	0	0	Egop_View.Appl_All_Id.Make_Prime_Key
0	1	0	0	Egop_View.Appl_All_Id.Initialized
0	31	0	3	Avzgope5_Aspect.Process_Steps.Edit_Top_Fields.Calculate_Un
2	55	0	16	Avzgope5_Aspect.Process_Steps.Edit_Top_Fields.Validate_Top_Inputs
0	11	0	0	Egop_View.Appl_Eor_Tbl.Make_Prime_Key
0	1	0	0	Egop_View.Appl_Eor_Tbl.Initialized
0	12	0	0	Egop_View.Appl_Atg_Month.Make_End_Dt_Key
0	3	0	0	Egop_View.Appl_Atg_Month.Initialized
1	83	0	1	Avzgope5_Aspect.Process_Steps.Edit_Top_Fields.Check_For_Obg
0	31	0	0	Egop_View.Appl_Exec_Cmt_Obg.Make_Obg_Doc_No_Key
0	9	0	0	Egop_View.Appl_Exec_Cmt_Obg.Initialized
0	63	0	27	Avzgope5_Aspect.Process_Steps.Edit_Bottom_Fields_And_Update
0	15	0	0	Egop_View.Appl_Rmb_Ord.Make_Prime_Key
1	42	0	0	Egop_View.Modify_Record
0	3	0	0	Egop_View.Appl_Lmt_Mjo.Initialized
0	15	0	0	Egop_View.Appl_Limit_Code_Tbl.Make_Prime_Key
0	1	0	0	Egop_View.Appl_Limit_Code_Tbl.Initialized
0	19	0	0	Egop_View.Appl_Lmt_Mjo.Make_Prime_Key
0	3	0	0	Egop_View.Appl_Exec_Pst.Initialized
0	16	0	0	Egop_View.Appl_All_Lgr.Make_Prime_Key
0	3	0	0	Egop_View.Appl_All_Lgr.Initialized
0	9	0	0	Egop_View.Appl_Exec_Eor_Tran.Initialized
1	42	0	0	Egop_View.Add_Record
0	11	0	0	Egop_View.Appl_Exec_Pst.Make_Prime_Key
0	5	0	0	Egop_View.Appl_Exec_Rpt.Initialized
0	65	0	0	Egop_View.Appl_Exec_Rpt.Make_Prime_Key
0	11	0	0	Egop_View.Appl_Cst_Cen.Make_Prime_Key
0	1	0	0	Egop_View.Appl_Cst_Cen.Initialized
0	7	0	0	Egop_View.Appl_Mdf_Tbl.Make_Prime_Key
0	1	0	0	Egop_View.Appl_Mdf_Tbl.Initialized
0	19	0	0	Egop_View.Appl_Bat_Out_Hdr.Make_Sta_Key
0	7	0	0	Egop_View.Appl_Bat_Out_Hdr.Initialized
0	3	0	0	Egop_View.Appl_Bat_Out_Dtl.Initialized
0	21	0	0	Egop_View.Appl_Bat_Out_Dtl.Make_Prime_Key
0	0	0	0	Avzgope5_Aspect.No_Local_Process

Medium 5

CC	DSM	I/O	ERRS	Module Name
7	12	0	0	Avzp9pe3_Aspect
0	8	0	1	Avzp9pe3_Aspect.Set_Initial_Screen
7	8	0	3	Avzp9pe3_Aspect.Process_Steps
6	113	0	2	Avzp9pe3_Aspect.Process_Steps.Edit_And_Update
0	3	0	0	Ep9p_View_Appl_Fxd_Ast_Initialized_e
0	22	0	1	Avzp9pe3_Aspect.Process_Steps.Edit_And_Update.Check_Requir
1	32	0	1	Avzp9pe3_Aspect.Process_Steps.Edit_And_Update.Calculate_Cp
0	3	0	0	Ep9p_View.Appl_Fxd_Ast_Amt.Initialized
0	15	0	0	Ep9p_View.Appl_Fxd_Ast_Amt.Make_Prime_Key
3	65	0	0	Ep9p_View.Retrieve
2	0	0	0	Ep9p_View.Operation_Status
0	3	0	0	Ep9p_View.Appl_Atg_Month.Initialized
0	12	0	0	Ep9p_View.Appl_Atg_Month.Make_End_Dt_Key
1	54	0	1	Avzp9pe3_Aspect.Process_Steps.Edit_And_Update.Update_Databa
1	34	0	0	Ep9p_View.Modify_Record
0	20	0	0	Ep9p_View.Appl_Ppy_Lgr.Make_Prime_Key
0	3	0	0	Ep9p_View.Appl_Ppy_Lgr.Initialized
0	3	0	0	Ep9p_View.Appl_Fa_Pst.Initialized
0	11	0	0	Ep9p_View.Appl_Fa_Pst.Make_Prime_Key
2	12	0	0	Ep9p_View.Release_Record
0	11	0	0	Ep9p_View.Appl_Fxd_Ast.Make_Prime_Key
0	0	0	0	Avzp9pe3_Aspect.No_Local_Process

Medium 6

CC	DSM	I/O	ERRS	Module Name
7	12	0	0	Avzntpe3_Aspect
1	64	0	2	Avzntpe3_Aspect.Set_Locks
1	7	0	1	Avzntpe3_Aspect.Set_Initial_Screen
0	27	0	3	Avzntpe3_Aspect.Process_Steps
2	48	0	3	Avzntpe3_Aspect.Process_Steps.Display_Fxd_Ast_Rec
0	3	0	0	Entp_View.Appl_Fxd_Ast.Initialized
0	61	0	1	Avzntpe3_Aspect.Process_Steps.Display_Fxd_Ast_Rec.Fill_Scrn
0	11	0	0	Entp_View.Appl_Fxd_Ast.Make_Prime_Key
3	68	0	0	Entp_View.Retrieve
2	0	0	0	Entp_View.Operation_Status
4	105	0	4	Avzntpe3_Aspect.Process_Steps.Edit_And_Update
0	35	0	0	Avzntpe3_Aspect.Process_Steps.Edit_And_Update.Field_Entered
0	15	0	1	Avzntpe3_Aspect.Process_Steps.Edit_And_Update.Dpr_Idc_Change
1	31	0	1	Avzntpe3_Aspect.Process_Steps.Edit_And_Update.Cp_Ast_Use_O~
1	31	0	0	Avzntpe3_Aspect.Process_Steps.Edit_And_Update.Cp_Ast_Acq_O~
0	21	0	2	Avzntpe3_Aspect.Process_Steps.Edit_And_Update.Field_Changed
0	71	0	47	Avzntpe3_Aspect.Process_Steps.Edit_And_Update.Edit_Fields_En
0	1	0	0	Entp_View.Appl_Cp_Ast_Nme_Tbl.Initialized
0	11	0	0	Entp_View.Appl_Cp_Ast_Nme_Tbl.Make_Prime_Key
3	35	0	0	Entp_View.Read_Index_Only
0	1	0	0	EntmO.Get_Cp_Ast_Acq_Own
0	1	0	0	Entp_View.Appl_Fsc_Nme_Tbl.Initialized
0	1	0	0	Entp_View.Appl_Fsc_Ast_~bl.Initialized
0	19	0	0	Entp_View.Appl_Fsc_Ast_~bl.Make_Prime_Key
0	15	0	0	Entp_View.Appl_Fsc_Nme_Tbl.Make_Prime_Key
0	1	0	0	Entp_View.Appl_Fac_Cat_~yp_Tbl.Initialized
0	1	0	0	Entp_View.Appl_Fac_Cat_~st_Tbl.Initialized
0	19	0	0	Entp_View.Appl_Fac_Cat_~st_Tbl.Make_Prime_Key
0	7	0	0	Entp_View.Appl_Fac_Cat_~yp_Tbl.Make_Prime_Key
0	177	0	13	Avzntpe3_Aspect.Process_Steps.Edit_And_Update.Build_Fxd_Ast
0	16	0	5	Avzntpe3_Aspect.Process_Steps.Edit_And_Update.Update_Genera
0	3	0	0	Entp_View.Appl_Fa_Pst.Initialized
0	20	0	0	Entp_View.Appl_Ppy_Lgr.Make_Prime_Key
0	3	0	0	Entp_View.Appl_Ppy_Lgr.Initialized
0	11	0	0	Entp_View.Appl_Fa_Pst.Make_Prime_Key
1	37	0	0	Entp_View.Modify_Record
2	15	0	0	Entp_View.Release_Record
0	18	0	0	Avzntpe3_Aspect.Process_Steps.Edit_And_Update.Reset_Screen
0	0	0	0	Avzntpe3_Aspect.No_Local_Process

Medium 7

CC	DSM	I/O	ERRS	Module Name
6	3	0	0	Avzcupm5_Aspect
0	17	0	13	Avzcupm5_Aspect.Initialize_Edit_Report
1	13	0	2	Avzcupm5_Aspect.Put_Report_Header
0	6	0	1	Avzcupm5_Aspect.Convert_To_Sjo
0	122	0	71	Avzcupm5_Aspect.Edit_Data
1	120	0	28	Avzcupm5_Aspect.Edit_Data.Determine_Atg_Month_Info
0	12	0	0	Mcup_View.Appl_Atg_Month.Make_End_Dt_Key
0	3	0	0	Mcup_View.Appl_Atg_Month.Initialized
0	1	0	0	Mcup_View.Appl_Ga_Sys_Isl.Initialized
0	11	0	0	Mcup_View.Appl_Ga_Sys_Isl.Make_Prime_Key
3	80	0	0	Mcup_View.Retrieve
2	0	0	0	Mcup_View.Operation_Status
2	99	0	14	Avzcupm5_Aspect.Edit_Data.Determine_Detail_String_Errors
0	91	0	38	Avzcupm5_Aspect.Edit_Data.Validate_Detail_Fields
7	206	0	31	Avzcupm5_Aspect.Edit_Data.Verify_Database_Records
0	1	0	0	Mcup_View.Appl_All_Id.Initialized
0	12	0	0	Mcup_View.Appl_Ast_Obj_Cd_Tbl.Make_Prime_Key
0	1	0	0	Mcup_View.Appl_Ast_Obj_Cd_Tbl.Initialized
0	3	0	0	Mcup_View.Appl_Eor_Tbl.Initialized
0	7	0	0	Mcup_View.Appl_Fnd_Mjo.Initialized
0	15	0	0	Mcup_View.Appl_Fnd_Sjo.Make_Prime_Key
0	1	0	0	Mcup_View.Appl_Fnd_Sjo.Initialized
0	15	0	0	Mcup_View.Appl_Rmb_Ord.Make_Prime_Key
0	5	0	0	Mcup_View.Appl_Rmb_Ord.Initialized

0	11	0	0	Mcup_View.Appl_Eor_Tbl.Make_Prime_Key
2	67	0	0	Mcup_View.Read_Index_Only
0	11	0	0	Mcup_View.Appl_All_Id.Make_Prime_Key
0	15	0	0	Mcup_View.Appl_Fnd_M_jo.Make_Prime_Key
0	41	0	27	Avzcupm5_Aspect.Update_Files
5	161	0	28	Avzcupm5_Aspect.Update_Files.Build_Detail_Record_Info
4	163	0	31	Avzcupm5_Aspect.Update_Files.Update_Database_Records
0	11	0	0	Mcup_View.Appl_Exec_Cmt_Obg.Initialized
0	27	0	0	Mcup_View.Appl_Exec_Cmt_Obg.Make_Prime_Key
0	5	0	0	Mcup_View.Appl_Lmt_Mjo.Initialized
0	44	0	9	Avzcupm5_Aspect.Update_Files.Update_Database_Records.Print_Up
1	64	0	10	Avzcupm5_Aspect.Update_Files.Update_Database_Records.Build_L~
0	1	0	0	Mcup_View.Appl_Limit_Code_Tbl.Initialized
0	15	0	0	Mcup_View.Appl_Limit_Code_Tbl.Make_Prime_Key
0	19	0	0	Mcup_View.Appl_Lmt=Mjo.Make_Prime_Key
0	1	0	0	Mcup_View.Appl_Cus_Typ_Tbl.Initialized
0	1	0	0	Mcup_View.Appl_Exe=Grp_Asg.Initialized
0	7	0	0	Mcup_View.Appl_Cus_Typ_Tbl.Make_Prime_Key
0	14	0	0	Mcup_View.Appl_Exec_Grp_Asg.Make_Prime_Key
0	1	0	0	Mcup_View.Record_Is_Locked
0	16	0	0	Mcup_View.Appl_All_Lgr.Make_Prime_Key
0	3	0	0	Mcup_View.Appl_All_Lgr.Initialized
0	11	0	0	Mcup_View.Appl_Exec_Pst.Make_Prime_Key
0	3	0	0	Mcup_View.Appl_Exec_Pst.Initialized
0	9	0	0	Mcup_View.Appl_Exec_Eor_Tran.Initialized
1	44	0	0	Mcup_View.Add_Record
1	44	0	0	Mcup_View.Modify_Record
1	69	0	10	Avzcupm5_Aspect.Update_Files.Update_Database_Records.Add_Dsc
1	83	0	20	Avzcupm5_Aspect.Update_Files.Update_Database_Records.Add_Tb
0	5	0	0	Mcup_View.Appl_Exec_Rpt.Initialized
0	65	0	0	Mcup_View.Appl_Exec_Rpt.Make_Prime_Key
0	26	0	0	Avzcupm5_Aspect.Update_Files.Fix_Errors
0	19	0	7	Avzcupm5_Aspect.Close_Edit_Report

Hard Programs

Hard 1

CC	DSM	I/O	ERRS	Module Name
4	5	0	0	Avzxypg8_Aspect
0	15	0	5	Avzxypg8_Aspect.Pss_Parm.Put
0	35	0	15	Avzxypg8_Aspect.Do_Edits
0	13	0	2	Avzxypg8_Aspect.Do_Edits.Set_Required_Field_Attr
1	55	0	12	Avzxypg8_Aspect.Do_Edits.Validate
0	3	0	0	Gxyp_View.Appl_Atg_Month.Initialized
0	15	0	0	Gxyp_View.Appl_Atg_Month.Make_Prime_Key
3	63	0	0	Gxyp_View.Retrieve
2	0	0	0	Gxyp_View.Operation_Status
0	45	0	2	Avzxypg8_Aspect.Do_Edits.Set_Invalid_Field_Attr
0	9	0	2	Avzxypg8_Aspect.Finish_Building_Record
1	7	0	1	Avzxypg8_Aspect.Lock_Fields

Hard 2

CC	DSM	I/O	ERRS	Module Name
7	12	0	0	Avzuepe4_Aspect
1	7	0	1	Avzuepe4_Aspect.Set_Initial_Screen
0	33	0	2	Avzuepe4_Aspect.Process_Steps
0	8	0	1	Avzuepe4_Aspect.Process_Steps.Request_Pending
12	38	0	4	Avzuepe4_Aspect.Process_Steps.Prepare_For_Step_2
4	181	0	8	Avzuepe4_Aspect.Process_Steps.Edit_Key_Data
0	5	0	0	Eucp_View.Appl_Fnd_Mjo.Initialized
0	3	0	0	Eucp_View.Appl_All_Id.Initialized
0	3	0	0	Eucp_View.Appl_Mjo_Dst.Initialized
0	30	0	0	Avzuepe4_Aspect.Process_Steps.Edit_Key_Data.Check_Step_1_In
0	16	0	1	Avzuepe4_Aspect.Process_Steps.Edit_Key_Data.Set_Error_Messag
3	28	0	5	Avzuepe4_Aspect.Process_Steps.Edit_Key_Data.Valid_Sys_Ba_Jo
0	3	0	0	Eucp_View.Appl_Fnd_Cntl_Ba.Initialized
0	23	0	0	Eucp_View.Appl_Fnd_Cntl_Ba.Make_Ba_All_Key
3	52	0	0	Eucp_View.Read_Index_Only
2	0	0	0	Eucp_View.Operation_Status
0	15	0	0	Eucp_View.Appl_Fnd_M_jo.Make_Prime_Key
3	66	0	0	Eucp_View.Retrieve
0	23	0	0	Eucp_View.Appl_All_Id.Make_Prime_Key
0	40	0	2	Avzuepe4_Aspect.Process_Steps.Validate_Screen_Amounts
0	12	0	0	Avzuepe4_Aspect.Process_Steps.Prepare_For_Step_3
1	39	0	2	Avzuepe4_Aspect.Process_Steps.Edit_Action_Data
6	142	0	5	Avzuepe4_Aspect.Process_Steps.Update_Action_Data
0	11	0	0	Eucp_View.Appl_Ga_Sys_Isl.Make_Prime_Key
0	3	0	0	Eucp_View.Appl_Ga_Sys_Isl.Initialized
0	54	0	1	Avzuepe4_Aspect.Process_Steps.Update_Action_Data.Check_Pass
1	75	0	0	Avzuepe4_Aspect.Process_Steps.Update_Action_Data.Prepare_For
0	144	0	5	Avzuepe4_Aspect.Process_Steps.Update_Action_Data.Get_Mjo_Re
0	76	0	1	Avzuepe4_Aspect.Process_Steps.Update_Action_Data.Determine
1	109	0	10	Avzuepe4_Aspect.Process_Steps.Update_Action_Data.Update_Dat
0	1	0	0	Eucp_View.Appl_Fnd_Ntv.Initialized
1	30	0	0	Eucp_View.Add_Record
1	35	0	0	Eucp_View.Modify_Record
1	9	0	2	Avzuepe4_Aspect.Process_Steps.Update_Action_Data.Prepare_Fo
0	0	0	0	Avzuepe4_Aspect.No_Local_Process

Hard 3

CC	DSM	I/O	ERRS	Module Name
6	3	0	0	Avzbfpm7_Aspect
0	22	0	15	Avzbfpm7_Aspect.Initialize_Edit_Report
0	17	0	2	Avzbfpm7_Aspect.Put_Report_Header

0	193	0	45	Avzbfpm7_Aspect.Edit_Data
1	29	0	4	Avzbfpm7_Aspect.Edit_Data.Get_Current_Atg_Month
0	12	0	0	Mbfp_View.Appl_Atg_Month.Make_End_Dt_Key
0	3	0	0	Mbfp_View.Appl_Atg_Month.Initialized
3	90	0	0	Mbfp_View.Retrieve
2	0	0	0	Mbfp_View.Operation_Status
0	165	0	21	Avzbfpm7_Aspect.Edit_Data.Determine_Fields_In_Error
0	32	0	14	Avzbfpm7_Aspect.Edit_Data.Check_For_Valid_Values
0	43	0	12	Avzbfpm7_Aspect.Edit_Data.Check_For_Valid_Values.Check_For
1	16	0	5	Avzbfpm7_Aspect.Edit_Data.Check_For_Valid_Values.Valid_Eor
0	11	0	0	Mbfp_View.Appl_Eor_Tbl.Make_Prime_Key
0	1	0	0	Mbfp_View.Appl_Eor_Tbl.Initialized
3	77	0	0	Mbfp_View.Read_Index_Only
0	15	0	0	Mbfp_View.Appl_Fnd_Sjo.Make_Prime_Key
0	1	0	0	Mbfp_View.Appl_Fnd_Sjo.Initialized
0	1	0	0	Mbfp_View.Appl_All_Id.Initialized
0	11	0	0	Mbfp_View.Appl_All_Id.Make_Prime_Key
1	36	0	3	Avzbfpm7_Aspect.Edit_Data.Check_For_Valid_Values.Jo_Valid_O
0	15	0	0	Mbfp_View.Appl_Misc_Sjo.Make_Prime_Key
0	3	0	0	Mbfp_View.Appl_Misc_Sjo.Initialized
1	16	0	8	Avzbfpm7_Aspect.Edit_Data.Check_For_Valid_Values.Valid_Cst_C
0	11	0	0	Mbfp_View.Appl_Cst_Cen.Make_Prime_Key
0	1	0	0	Mbfp_View.Appl_Cst_Cen.Initialized
1	14	0	0	Avzbfpm7_Aspect.Edit_Data.Check_For_Valid_Values.Valid_Dssn
0	7	0	0	Mbfp_View.Appl_Dssn_Un_Tbl.Make_Prime_Key
0	1	0	0	Mbfp_View.Appl_Dssn_Un_Tbl.Initialized
2	67	0	34	Avzbfpm7_Aspect.Edit_Data.Check_For_Valid_Values.Valid_Sys_B
0	19	0	0	Mbfp_View.Appl_Exec_Bll.Make_Sjo_Bll_Key
0	5	0	0	Mbfp_View.Appl_Exec_Bll.Initialized
0	3	0	0	Mbfp_View.Appl_Dssn_Adv_Cntl.Initialized
0	15	0	0	Mbfp_View.Appl_Dssn_Adv_Cntl.Make_Prime_Key
0	36	0	0	Avzbfpm7_Aspect.Edit_Data.Check_For_Valid_Values.Valid_Mat
0	101	0	34	Avzbfpm7_Aspect.Edit_Data.Check_For_Valid_Values.Validate_Ag
1	21	0	0	Avzbfpm7_Aspect.Edit_Data.Get_Todays_Xmt_No
0	8	0	0	Mbfp_View.Appl_Xmt_No_Tbl.Make_Prime_Key
0	1	0	0	Mbfp_View.Appl_Xmt_No_Tbl.Initialized
0	129	0	52	Avzbfpm7_Aspect.Update_Files
1	35	0	13	Avzbfpm7_Aspect.Update_Files.Get_Atg_Month_Rec
0	9	0	0	Avzbfpm7_Aspect.Update_Files.Make_Atg_Dt
1	22	0	7	Avzbfpm7_Aspect.Update_Files.Get_Fnd_Sjo_Rec
0	163	0	46	Avzbfpm7_Aspect.Update_Files.Determine_Trans_Typ
0	15	0	0	Mbfp_View.Appl_Fnd_Mjo.Make_Prime_Key
0	1	0	0	Mbfp_View.Appl_Fnd_Mjo.Initialized
0	11	0	0	Mbfp_View.Appl_Rmb_Ord.Make_Prime_Key
0	1	0	0	Mbfp_View.Appl_Rmb_Ord.Initialized
1	19	0	0	Avzbfpm7_Aspect.Update_Files.Retrieve_Do_Pst
0	7	0	0	Mbfp_View.Appl_Do_Pst.Make_Prime_Key
0	1	0	0	Mbfp_View.Appl_Do_Pst.Initialized
2	78	0	14	Avzbfpm7_Aspect.Update_Files.Create_Dssn_Doc_0th
0	9	0	0	Mbfp_View.Appl_Dssn_Doc_Oth.Initialized
1	54	0	0	Mbfp_View.Add_Record
1	64	0	27	Avzbfpm7_Aspect.Update_Files.Create_Dssn_Doc_Slf
1	106	0	8	Avzbfpm7_Aspect.Update_Files.Create_Dssn_302t_Rpt
0	3	0	0	Mbfp_View.Appl_Dssn_302t_Rpt.Initialized
1	237	0	53	Avzbfpm7_Aspect.Update_Files.Update_Collection
0	22	0	14	Avzbfpm7_Aspect.Update_Files.Update_Collection.Get_Exec_Bll
0	26	0	5	Avzbfpm7_Aspect.Update_Files.Update_Collection.Calculate_Dist
1	56	0	27	Avzbfpm7_Aspect.Update_Files.Update_Collection.Update_Rmb_Or
1	54	0	0	Mbfp_View.Modify_Record
3	208	0	12	Avzbfpm7_Aspect.Update_Files.Update_Collection.Update_All_Lg
0	3	0	0	Mbfp_View.Appl_Ar_Pst.Initialized
0	1	6	0	Mbfp_View.Appl_All_Lgr.Make_Prime_Key
0	3	0	0	Mbfp_View.Appl_All_Lgr.Initialized
0	9	0	0	Mbfp_View.Appl_Exec_Eor_Tran.Initialized
0	11	0	0	Mbfp_View.Appl_Ar_Pst.Make_Prime_Key
0	194	0	66	Avzbfpm7_Aspect.Update_Files.Update_Collection.Update_All_Ar
0	33	0	0	Mbfp_View.Appl_Ar_Rpt_Ipa.Make_Prime_Key
0	5	0	0	Mbfp_View.Appl_Ar_Rpt_Ipa_Initialized
1	52	0	18	Avzbfpm7_Aspect.Update_Files.Update_Collection.Update_Dssn_A
0	19	0	0	Mbfp_View.Appl_Exec_Rpt.Make_Prime_Key
0	1	0	0	Mbfp_View.Appl_Exec_Rpt.Initialized

1	85	0	10	Avzbfpm7_Aspect.Update_Files.Update_Rpt_239_Rec
0	55	0	0	Mbfp_View.Appl_Rpt_239.Make_Prime_Key
0	3	0	0	Mbfp_View.Appl_Rpt_239.Initialized
4	180	0	48	Avzbfpm7_Aspect.Update_Files.Update_Dssn_Day_Sum
0	3	0	0	Mbfp_View.Appl_Dssn_Day_Sum.Initialized
0	36	0	0	Mbfp_View.Appl_Dssn_Day_Sum.Make_Prime_Key
0	7	0	1	Avzbfpm7_Aspect.Update_Files.Calculate_Mo_Of_Qtr
0	18	0	0	Avzbfpm7_Aspect.Close_Edit_Report
0	5	0	6	Cv_Edit_Rpt.Put_Control_Total

Hard 4

CC	DSM	I/O	ERRS	Module Name
7	12	0	0	Avzglpe5_Aspect
0	13	0	4	Avzglpe5_Aspect.Set_Initial_Screen
2	102	0	21	Avzglpe5_Aspect.Process_Steps
0	59	0	8	Avzglpe5_Aspect.Process_Steps.Set_Top_Attr
0	82	0	26	Avzglpe5_Aspect.Process_Steps.Set_Bottom_Attr
2	69	0	2	Avzglpe5_Aspect.Process_Steps.Retrieve_Fnd_Sjo_Rec
0	15	0	0	Eglp_View.Appl_Fnd_Sjo.Make_Prime_Key
0	1	0	0	Eglp_View.Appl_Fnd_S_jo.Initialized
3	88	0	0	Eglp_View.Retrieve
2	0	0	0	Eglp_View.Operation_Status
1	21	0	0	Avzglpe5_Aspect.Process_Steps.Get_Fnd_M_jo_Rec
0	9	0	0	Eglp_View.Appl_Fnd_Mjo.Make_Prime_Key
0	15	0	0	Eglp_View.Appl_Fnd_Mjo.Initialized
2	42	0	84	Avzglpe5_Aspect.Process_Steps.Edit_Top_Of_Screen
0	120	0	14	Avzglpe5_Aspect.Process_Steps.Edit_Top_Of_Screen.Check_Requi
3	85	0	30	Avzglpe5_Aspect.Process_Steps.Edit_Top_Of_Screen.Validate_In
0	3	0	0	Eglp_View.Appl_Eor_Tbl.Initialized
0	11	0	0	Eglp_View.Appl_Eor_Tbl.Make_Prime_Key
0	1	0	0	Eglp_View.Appl_Cce_Jo_Corr_Tbl.Initialized
0	11	0	0	Eglp_View.Appl_Cce_Jo_Corr_Tbl.Make_Prime_Key
3	75	0	0	Eglp_View.Read_Index_Only
0	20	0	0	Eglp_View.Appl_Ppy_Lgr.Make_Prime_Key
0	3	0	0	Eglp_View.Appl_Ppy_Lgr.Initialized
0	12	0	0	Eglp_View.Appl_Atg_Month.Make_End_Dt_Key
0	3	0	0	Eglp_View.Appl_Atg_Month.Initialized
2	99	0	12	Avzglpe5_Aspect.Process_Steps.Edit_Top_Of_Screen.Validate_Pr
0	11	0	20	Eglp_View.Appl_Exec_Cmt_Obg.Initialized
0	31	0	0	Eglp_View.Appl_Exec_Cmt_Obg.Make_Obg_Doc_No_Key
0	3	0	0	Eglp_View.Appl_Fxd_Ast.Initialized
0	15	0	0	Eglp_View.Appl_Fxd_Ast.Make_Prime_Key
1	59	0	25	Avzglpe5_Aspect.Process_Steps.Edit_Top_Of_Screen.Build_O/P_S
0	1	0	0	Eglp_View.Appl_All_Id.Initialized
0	11	0	0	Eglp_View.Appl_All_Id.Make_Prime_Key
14	338	0	6	Avzglpe5_Aspect.Process_Steps.'Update_Records
0	3	0	0	Eglp_View.Appl_All_Lgr.Initialized
0	5	0	0	Eglp_View.Appl_Exec_Rpt.Initialized
0	15	0	0	Eglp_View.Appl_Rmb_Ord.Make_Prime_Key
0	5	0	0	Eglp_View.Appl_Rmb_Ord.Initialized
7	406	0	7	Avzglpe5_Aspect.Process_Steps.Update_Records.Update_Fs_Recs
0	27	0	0	Eglp_View.Appl_Exec_Cmt_Obg.Make_Prime_Key
0	19	0	0	Eglp_View.Appl_Limit_Code_Tbl.Make_Prime_Key
0	3	0	0	Eglp_View.Appl_Limit_Code_Tbl.Initialized
1	52	0	1	Eglp_View.Modify_Record
0	3	0	0	Eglp_View.Appl_Exec_Pst.Initialized
0	11	0	0	Eglp_View.Appl_Exec_Pst.Make_Prime_Key
0	11	0	0	Eglp_View.Appl_Exec_Eor_Tran.Initialized
1	52	0	0	Eglp_View.Add_Record
0	1	0	0	Eglp_View.Appl_Cus_Typ_Tbl.Initialized
0	7	0	0	Eglp_View.Appl_Cus_Typ_Tbl.Make_Prime_Key
0	1	0	0	Eglp_View.Appl_Exec_Grp_Asg.Initialized
0	19	0	0	Eglp_View.Appl_Exec_Grp_Asg.Make_Prime_Key
0	16	0	0	Eglp_View.Appl_All_Lgr.Make_Prime_Key
2	35	0	0	Eglp_View.Release_Record
0	65	0	0	Eglp_View.Appl_Exec_Rpt.Make_Prime_Key
0	1	0	0	Eglp_View.Record_Is_Locked
4	102	0	0	Avzglpe5_Aspect.Process_Steps.Update_Fa_Recs

0	11	0	0	Eglp_View.Appl_Ga_Sys_Isl.Make_Prime_Key
0	3	0	0	Eglp_View.Appl_Ga_Sys_Isl.Initialized
0	6	0	0	Avzglpe5_Aspect.Process_Steps.Update_Fa_Recs.Get_Cp_Amt_Cd
1	21	0	5	Avzglpe5_Aspect.Process_Steps.Update_Fa_Recs.Update_Faa_Rec
0	3	0	0	Eglp_View.Appl_Fxd_Ast_Amt.Initialized
1	124	0	48	Avzglpe5_Aspect.Process_Steps.Update_Fa_Recs.Add_Fxd_Ast_Re
2	102	0	0	Avzglpe5_Aspect.Process_Steps.Update_Fa_Recs.Update_Ppy_Lgr
0	3	0	0	Eglp_View.Appl_Fa_Pst.Make_Prime_Key
0	11	0	0	Eglp_View.Appl_Fa_Pst.Initialized
2	56	0	0	Avzglpe5_Aspect.Process_Steps.Edit_Bottom_And_Update
1	61	0	14	Avzglpe5_Aspect.Process_Steps.Edit_Bottom_And_Update.Check
0	1	0	0	Eglp_View.Appl_Fsc_Ast_Tbl.Initialized
0	11	0	0	Eglp_View.Appl_Fsc_Nme_Tbl.Make_Prime_Key
0	3	0	0	Eglp_View.Appl_Fsc_Nme_Tbl.Initialized
0	19	0	0	Eglp_View.Appl_Fsc_Ast_Tbl.Make_Prime_Key
0	1	0	0	Eglp_View.Appl_Fac_Cat_Ast_Tbl.Initialized
0	3	0	0	Eglp_View.Appl_Fac_Cat_Typ_Tbl.Initialized
0	7	0	0	Eglp_View.Appl_Fac_Cat_Typ_Tbl.Make_Prime_Key
0	19	0	0	Eglp_View.Appl_Fac_Cat_Ast_Tbl.Make_Prime_Key
0	7	0	0	Eglp_View.Appl_Mdf_Tbl.Make_Prime_Key
0	1	0	0	Eglp_View.Appl_Mdf_Tbl.Initialized
0	0	0	2	Avzglpe5_Aspect.No_Local_Process

Hard 5

CC	DSM	I/O	ERRS	Module Name
7	12	0	0	Avzmupe5_Aspect
0	108	0	7	Avzmupe5_Aspect.Set_Key_Fields
0	28	0	4	Avzmupe5_Aspect.Set_Initial_Screen
0	36	0	3	Avzmupe5_Aspect.Process_Steps
0	269	0	8	Avzmupe5_Aspect.Process_Steps.Edit
0	17	0	0	Avzmupe5_Aspect.Process_Steps.Edit.Validate_Fic
1	49	0	1	Avzmupe5_Aspect.Process_Steps.Edit.Validate_Eor
0	3	0	0	Emup_View.Appl_Eor_Tbl.Initialized
0	11	0	0	Emup_View.Appl_Eor_Tbl.Make_Prime_Key
3	87	0	0	Emup_View.Retrieve
2	0	0	0	Emup_View.Operation_Status
0	20	0	1	Avzmupe5_Aspect.Process_Steps.Edit.Validate_Odc
0	30	0	2	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Odc
0	68	0	4	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Required_Fields
1	40	0	5	Avzmupe5_Aspect.Process_Steps.Edit.Get_Misc_S_jo
0	7	0	0	Emup_View.Appl_Misc_Sjo.Initialized
0	27	0	0	Emup_View.Appl_Misc_S_jo.Make_Prime_Key
0	229	0	11	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Funded
1	38	0	0	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Funded.Get_Fnd_Sjo
0	1	0	0	Emup_View.Appl_Fnd_Sjo.Initialized
0	15	0	0	Emup_View.Appl_Fnd_Sjo.Make_Prime_Key
0	30	0	1	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Funded.Required_Fund
1	38	0	0	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Funded.Validate_Bu_V
0	1	0	0	Emup_View.Appl_Bu_Vou_Def.Initialized
0	11	0	0	Emup_View.Appl_Bu_Vou_Def.Make_Bu_Vou_Key
1	53	0	0	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Funded.Validate_Cst_C
0	1	0	0	Emup_View.Appl_Ccc_Jo_Corr_Tbl.Initialized
0	23	0	0	Emup_View.Appl_Ccc_Jo_Corr_Tbl.Make_Prime_Kev
1	56	0	3	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Funded.Get_Pvrl_Cmt
1	73	0	5	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Funded.Get_Exec_Cmt
0	11	0	0	Emup_View.Appl_Exec_Cmt_Obg.Initialized
0	31	0	0	Emup_View.Appl_Exec_Cmt_Obg.Make_Obg_Doc_No_Kev
1	70	0	3	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Funded.Get_Exec_BV_P
0	27	0	0	Emup_View.Appl_Exec_Cmt_Obg.Make_Prime_Kev
0	94	0	0	AvzmupeS_Aspect.Process_Steps.Edit.Edit_Tfo
0	26	0	1	Avzmupe5_Aspect.Process_Steps.Edit.Edit_Tfo.Required_Tfo_Fie
1	25	0	1	Avzmupe5_Aspect.Process_Steps.Edit.Validate_Dssn
0	1	0	0	Emup_View.Appl_Dssn_Un_Tbl.Initialized
0	7	0	0	Emup_View.Appl_Dssn_Un_Tbl.Make_Prime_Kev
0	223	0	0	Avzmupe5_Aspect.Process_Steps.Edit.Format_Screen
1	80	0	7	Avzmupe5_Aspect.Process_Steps.Edit.Get_Atg_Month
0	3	0	0	Emup_View_Appl_Atg_Month.Initialized
0	12	0	0	Emup_View.Appl_Atg_Month.Make_End_Dt_Kev

0	5	0	2	Avzmupe5_Aspect.Process_Steps.Edit.Get_Atg_Month.Calculate_M
0	10	0	1	Avzmupe5_Aspect.Process_Steps.Edit.Get_Atg_Month.Create_Atg
1	28	0	0	Avzmupe5_Aspect.Process_Steps.Edit.Get_Xmt_No_Tbl
0	653	0	48	Avzmupe5_Aspect.Process_Steps.Update
0	64	0	2	Avzmupe5_Aspect.Process_Steps.Update.Reset_To_Step
0	40	0	1	Avzmupe5_Aspect.Process_Steps.Update.Reset_To_Step_2
0	23	0	0	Avzmupe5_Aspect.Process_Steps.Update.Edit_Conf_irm
0	30	0	0	Avzmupe5_Aspect.Process_Steps.Update.Edit_Trf_Amount
0	28	0	0	Avzmupe5_Aspect.Process_Steps.Update.Edit_Trf_Hrs
1	40	0	0	Avzmupe5_Aspect.Process_Steps.Update.Get_Misc_S_jo
0	117	0	2	Avzmupe5_Aspect.Process_Steps.Update.Update_Funded
1	46	0	5	Avzmupe5_Aspect.Process_Steps.Update.Update_Funded.Get_Pst
0	1	0	0	Emup_View.Appl_Cus_Typ_Tbl.Initialized
0	1	0	0	Emup_View.Appl_Rmb_Ord.Initialized
0	8	0	0	Emup_View.Appl_Rmb_Ord.Make_Prime_Kev
0	7	0	0	Emup_View.Appl_Cus_Tvp_Tbl.Make_Prime_Kev
0	1	0	0	Emup_View.Appl_Exec_Grp_Asg.Initialized
0	19	0	0	Emup_View.Appl_Exec_Grp_Asg.Make_Prime_Kev
1	41	0	1	Avzmupe5_Aspect.Process_Steps.Update.Update_Funded.Locate_C
0	50	0	0	Avzmupe5_Aspect.Process_Steps.Update.Update_Funded.Load_Scrn
2	85	0	0	Avzmupe5_Aspect.Process_Steps.Update.Update_Funded.Load_Fnd
1	28	0	0	Avzmupe5_Aspect.Process_Steps.Update.Update_Funded.Load_Fnd
0	7	0	0	Emup_View.Appl_Fnd_Mjo.Initialized
0	15	0	0	Emup_View.Appl_Fnd_M_jo.Make_Prime_Kev
0	229	0	0	Avzmupe5_Aspect.Process_Steps.Update.Update_Funded.Edit_Fund
1	38	0	0	Avzmupe5_Aspect.Process_Steps.Update.Update_Funded.Load_Eor
1	26	0	0	Avzmupe5_Aspect.Process_Steps.Update.Update_Funded.Load_All
0	1	0	0	Emup_View.Appl_All_Id.Initialized
0	11	0	0	Emup_View.Appl_All_Id.Make_Prime_Kev
1	148	0	6	Avzmupe5_Aspect.Process_Steps.Update.Update_Funded.Update_Pv
1	51	0	4	Emup_View.Add_Record
1	51	0	0	Emup_View.Modifv_Record
2	34	0	0	Emup_View.Delete_Record
2	114	0	0	Avzmupe5_Aspect.Process_Steps.Update.Update_Tfo
1	30	0	4	Avzmupe5_Aspect.Process_Steps.Update.Update_Tfo.Get_Eor_Tbl
1	87	0	3	Avzmupe5_Aspect.Process_Steps.Update.Update_Tfo.Add_Dssn_30
0	3	0	0	Emup_View.Appl_Dssn_302t_Rpt.Initialized
1	122	0	0	Avzmupe5_Aspect.Process_Steps.Update.Update_Tfo.Add_Dssn_Do
0	13	0	0	Emup_View.Appl_Dssn_Doc_Oth.Initialized
1	127	0	7	Avzmupe5_Aspect.Process_Steps.Update.Update_Exec_Rpt
0	5	0	0	Emup_View.Appl_Exec_Rpt.Initialized
0	65	0	0	Emup_View.Appl_Exec_Rpt.Make_Prime_Key
2	75	0	2	Avzmupe5_Aspect.Process_Steps.Update.Update_Rmb_Ord
1	88	0	2	Avzmupe5_Aspect.Process_Steps.Update.Update_Dssn_302t_Rpt
1	55	0	3	Avzmupe5_Aspect.Process_Steps.Update.Update_Dssn_Doc_Slf
0	9	0	0	Emup_View.Appl_Dssn_Doc_Slf.Initialized
1	73	0	2	Avzmupe5_Aspect.Process_Steps.Update.Update_Fnd_M_jo
1	56	0	6	Avzmupe5_Aspect.Process_Steps.Update.Update_Lmt_M_jo
0	3	0	0	Emup_View.Appl_Lmt_Mjo.Initialized
0	3	0	0	Emup_View.Appl_Limit_Code_Tbl.Initialized
0	15	0	0	Emup_View.Appl_Limit_Code_Tbl.Make_Prime_Key
0	19	0	0	Emup_View.Appl_Lmt_Mjo.Make_Prime_Key
1	56	0	3	Avzmupe5_Aspect.Process_Steps.Update.Add_Exec_Eor_Tran
0	9	0	0	Emup_View.Appl_Exec_Eor_Tran.Initialized
2	100	0	7	Avzmupe5_Aspect.Process_Steps.Update.Update_General_Ledger
0	3	0	0	Emup_View.Appl_All_Lgr.Initialized
0	16	0	0	Emup_View.Appl_All_Lgr.Make_Prime_Key
0	3	0	0	Emup_View.Appl_Exec_Pst.Initialized
0	11	0	0	Emup_View.Appl_Exec_Pst.Make_Prime_Key
0	0	0	0	Avzmupe5_Aspect.No_Local_Process

Hard 6

CC	DSM	I/O	ERRS	Module Name
7	12	0	0	Avzorpe_1_Aspect
0	8	0	0	Avzorpe_1_Aspect.Set_Initial_Screen
0	21	0	0	Avzorpe_1_Aspect.Process_Steps
3	42	0	0	Avzorpe1_Aspect.Process_Steps.Edit_And_Update
0	1	0	0	Eorp_View.Appl_Svc_Mst_Tbl.Initialized

0	3	0	0	Eorp_View.Appl_Svc_Cus_Mst_Tbl.Initialized
9	246	0	21	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update
0	3	0	0	Eorp_View.Appl_Svc_Dst_Tbl.Initialized
0	3	0	0	Eorp_View.Appl_All_Lgr.Initialized
0	1	0	0	Eorp_View.Appl_Fnd_Sjo.Initialized
0	7	0	0	Eorp_View.Appl_Fnd_Mjo.Initialized
0	1	0	0	Eorp_View.Appl_Cus_Typ_Tbl.Initialized
2	123	0	3	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update.GI_Upd
0	3	0	0	Eorp_View.Appl_Exec_Pst.Initialized
0	11	0	0	Eorp_View.Appl_Exec_Pst.Make_Prime_Key
0	27	0	0	EormO.Get_Pst_Grp_Id
0	9	0	0	Eorp_View.Appl_Exec_Eor_Tran.Initialized
3	75	0	0	Eorp_View.Retrieve
2	0	0	0	Eorp_View.Operation_Status
0	1	0	0	Eorp_View.Appl_Exec_Grp_Asg.Initialized
0	19	0	0	Eorp_View.Appl_Exec_Grp_Asg.Make_Prime_Key
0	45	0	0	Avzorpe1_Aspect.Process_Steps.Edit_And_Update.Update.Prepare
0	8	0	0	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update.Modify
1	44	0	0	Eorp_View.Modify_Record
0	14	0	0	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update.Modify
0	33	0	0	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update.Retrie
0	15	0	0	Eorp_View.Appl_Fnd_Mjo.Make_Prime_Key
1	95	0	0	Avzorpe1_Aspect.Process_Steps.Edit_And_Update.Update.Update
1	58	0	0	Avzorpe1_Aspect.Process_Steps.Edit_And_Update.Update.Update
0	3	0	0	Eorp_View.Appl_Ar_Rpt.Initialized
0	31	0	0	Eorp_View.Appl_Ar_Rpt.Make_Prime_Key
1	103	0	0	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update.Update
0	23	0	0	Eorp_View.Appl_Exec_Bll.Initialized
0	41	0	0	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update.Update
0	5	0	0	Eorp_View.Appl_Exec_Rpt.Initialized
0	51	0	0	Avzorpe1_Aspect.Process_Steps.Edit_And_Update.Update.Update
0	65	0	0	Eorp_View.Appl_Exec_Rpt.Make_Prime_Key
0	77	0	5	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update.Do_Simp
0	188	0	9	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update.Do_Non
0	188	0	9	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update.Do_Non
0	16	0	0	Eorp_View.Appl_All_Lgr.Make_Prime_Key
0	7	0	0	Eorp_View.Appl_Cus_Typ_Tbl.Make_Prime_Key
0	15	0	0	Eorp_View.Appl_Fnd_Sjo.Make_Prime_Key
0	12	0	0	Eorp_View.Appl_Atg_Month.Make_End_Dt_Key
0	3	0	0	Eorp_View.Appl_Atg_Month.Initialized
2	76	0	8	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Update.Check_A
0	31	0	0	Eorp_View.Appl_Svc_Dst_Tbl.Make_Prime_Key
0	42	0	0	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Edit_Svc_Tvp
0	19	0	0	Eorp_View.Appl_Svc_Cus_Mst_Tbl.Make_Prime_Key
0	65	0	0	Avzorpe_1_Aspect.Process_Steps.Edit_And_Update.Edit_Svc_Tvp
0	15	0	0	Eorp_View.Appl_Svc_Mst_Tbl.Make_Prime_Key
0	0	0	0	Avzorpe_1_Aspect.No_Local_Process

Hard 7

CC	DSM	I/O	ERRS	Module Name
9	13	0	0	Avz67ph8_Aspect
0	27	0	0	H67p_View.Appl_Exec_Cmt_Obg.Make_Prime_Key
0	3	0	0	H67p_View.Appl_Exec_Cmt_Obg.Initialized
3	65	0	0	H67p_View.Read_Index_Only
2	0	0	0	H67p_View.Operation_Status
0	37	0	0	H67p_View.Appl_Fnd_Cmt_Req.Make_Prime_Key
0	1	0	0	H67p_View.Appl_Fnd_Cmt_Req.Initialized
0	15	0	0	H67p_View.Appl_Rmb_Ord.Make_Prime_Key
0	3	0	0	H67p_View.Appl_Rmb_Ord.Initialized
0	15	0	0	H67p_View.Appl_Fnd_Sjo.Make_Prime_Key
0	1	0	0	H67p_View.Appl_Fnd_Sjo.Initialized
0	195	0	2	Avz67ph8_Aspect.Build_Screen
0	13	0	5	Avz67ph8_Aspect.Process_Steps
0	103	0	14	Avz67ph8_Aspect.Process_Steps.Perform_Action
1	29	0	0	Avz67ph8_Aspect.Process_Steps.Perform_Action.Get_Rf_Idc
0	1	0	0	H67p_View.Appl_Bs_Edit_Tbl.Initialized
0	15	0	0	H67p_View.Appl_Bs_Edit_Tbl.Make_Prime_Key
3	78	0	0	H67p_View.Retrieve

1	39	0	3	Avz67ph8_Aspect.Process_Steps.Perform_Action.Add_Record
0	5	0	0	H67p_View.Appl_Jom_Tbl.Initialized
0	233	0	20	Avz67ph8_Aspect.Process_Steps.Perform_Action.Add_Record.Put
1	42	0	0	H67p_View.Add_Record
1	60	0	1	Avz67ph8_Aspect.Process_Steps.Perform_Action.Delete_Record
0	11	0	0	H67p_View.Appl_Jom_Tbl.Make_Prime_Key
2	25	0	0	H67p_View.Delete_Record
1	216	0	48	Avz67ph8_Aspect.Process_Steps.Perform_Action.Modify_Record
1	42	0	0	H67p_View.Modify_Record
2	25	0	0	H67p_View.Release_Record
0	206	0	14	Avz67ph8_Aspect.Process_Steps.Perform_Action.Edit_For_Inval
0	19	0	0	H67p_View.Appl_Jom_Tbl.Make_Jom_Mjo_Key
0	7	0	0	H67p_View.Appl_Ams_Tbl.Make_Prime_Key
0	1	0	0	H67p_View.Appl_Ams_Tbl.Initialized
0	1	0	0	H67p_View.Appl_Ams_Sys_Ttl_Tbl.Initialized
0	7	0	0	H67p_View.Appl_Ams_Sys_Ttl_Tbl.Make_Prime_Key
0	1	0	0	H67p_View.Appl_Ams_Sp_Prj_Tbl.Initialized
0	11	0	0	H67p_View.Appl_Ams_Sp_Prj_Tbl.Make_Prime_Key
0	1	0	0	H67p_View.Appl_Oh_Apl_Cd_Tbl.Initialized
0	7	0	0	H67p_View.Appl_Oh_Apl_Cd_Tbl.Make_Prime_Key
0	1	0	0	H67p_View.Appl_Cst_Cen.Initialized
0	11	0	0	H67p_View.Appl_Cst_Cen.Make_Prime_Key
0	1	0	0	H67p_View.Appl_Uic_Mst_Tbl.Initialized
0	11	0	0	H67p_View.Appl_Uic_Mst_Tbl.Make_Prime_Key
0	1	0	0	H67p_View.Appl_Hsc_Wt_Fer_Tbl.Initialized
0	21	0	0	H67p_View.Appl_Hsc_Wt_Fer_Tbl.Make_Prime_Key
0	1	0	0	H67p_View.Appl_Dir_Idr_Ttl_Tbl.Initialized
0	7	0	0	H67p_View.Appl_Dir_Idr_Ttl_Tbl.Make_Prime_Key
0	1	0	0	H67p_View.Appl_Std_Inx.Initialized
0	11	0	0	H67p_View.Appl_Std_Inx.Make_Prime_Key
0	4	0	0	Apn_Fy_Dsg_Pkg.Valid
0	9	0	0	Avz67ph8_Aspect.Process_Steps.Transfer_Control
3	167	0	16	Avz67ph8_Aspect.Receive_Control
0	1	0	0	H67p_View.Appl_All_Id.Initialized
0	1	0	0	H67p_View.Appl_Fnd_Mjo.Initialized
0	67	0	0	Avz67ph8_Aspect.Receive_Control.Display_M_jo_Fields
0	15	0	0	H67p_View.Appl_Fnd_Mjo.Make_Prime_Key
0	11	0	0	H67p_View.Appl_All_Id.Make_Prime_Key
0	0	0	0	Avz67ph8_Aspect.No_Local_Process
0	0	0	0	Avz67ph8_Aspect.No_Input_Process