ATACOBOL – A COBOL Test Coverage Analysis Tool and Its Applications

Sam K.S. Sze
Hong Kong and Shanghai Banking Corporation

Michael R. Lyu
Computer Science and Engineering Department
The Chinese University of Hong Kong

Abstract

A coverage testing tool ATACOBOL (Automatic Test Analysis for COBOL) that applies data flow coverage technique is developed for software development on IBM System/390 mainframe. We show that the data flow coverage criteria can identify possible problematic paths that maps to the actual testing semantic required by Y2K compliance software testing. However, the mainframe environment lacks testing tools that equip the data flow coverage measure. Up to the current implementation, ATACOBOL is able to perform block coverage, decision coverage and all-uses measures. We extend the rules of data flow coverage criteria to adapt data structures that modern high-level languages usually employ.

1. Introduction

Program testing is the most commonly used method for demonstrating that a program accomplishes its intended purpose [1]. It involves selecting elements from the program's input domain *D*, executing the program *P* on test cases *T*, and comparing the actual output with the expected output. On this base, we assume the existence of some methods to determine whether or not the output produced by a program is correct.

While testing all possible inputs values would provide the most complete picture of a program's behavior, the input domain is usually too large for exhaustive testing to be practical. From another point of view, *T* is generally associated with a set Π of paths through *P*'s flow graph. It means that we usually cannot exhaust all possible paths of the program *P*.

The usual procedure is to select a relatively small *T* which in some sense represent the entire *D* or implicitly all paths Π. Observation of the program on this subset is then used to predict its behavior in general. Unfortunately, discovering such an ideal set of test data is almost an impossible task [2].

A number of path selection criteria *C* have been proposed. The most well known criteria are the statement coverage and the edge coverage. Weyuker *et al* proposed a family of path selection criteria that include the control flow coverage criteria and an additional set of data flow selection criteria in terms of the def-use pairs [3], [4]. In their approach, variable occurrences are classified as being a definitional, computation-use or predicate-use occurrence. They are referred as def, c-use, and p-use, respectively. Furthermore,

- def: When a variable is assigned by a certain value.
- c-use : The variable is used in computation. It directly affects the computation being performed and may indirectly affect the flow of control through the program.
- p-use: The variable is used as a predicate to affect the flow of control through the program, and may indirectly affect the computations performed.
- def(*i*) is the set of variables for which node *i* contains a global def.
- c-use(*i*) is the set of variables for which node *i* contains a global c-use.
- p-use(*i, j*) is the set of variables for which edge (*i, j*) contains a p-use.
- dcu(*x, i*) is the set of all nodes *j* such that *x* belongs to c-use(*j*) and for which there is a def-clear path w.r.t. *x* from *i* to *j*.
- dpu(*x, i*) is the set of all edges (*j, k*) such that *x* belongs to p-use(*j, k*) and for which there is a def-clear path w.r.t. *x* from *i* to *j*.

2. A Y2K Testing Example that Requires Data Flow Coverage

This practical Y2K testing example is inspired from Year 2000 testing in the banking business. It illustrates how the all-uses criteria can be a stronger criteria than all-edges and capable to detect more faults in practice.

Some accounts opened with banks are temporary, for example, loan accounts. These temporary account records carry expiration dates. Before the manipulation of a temporary account, the account should be verified if it has been expired. Since most of the date fields in files carries only 2 digits, to cater for Y2K problem, one way is to apply the 49/50 rule to determine the century. This is shown in the program listed in Figure 1. The control flow graph of this program fragment is shown on Figure 2. The def/use graph is shown on Table 1.

```
/* Init expire indicator */
Exp_Indicator = 0;

* Read year of today's date and expire date */
read Today_Year;                                        b1
read Exp_Year;

/* Set up today's year using 49/50 Rule */
if (Today_Year < 50)
{
    Today_Year = 2000 + Today_Year;                     b2
}
  else
{
    Today_Year = 1900 + Today_Year;                     b3
}
/* Set up expire year using 49/50 Rule and hence check if expired */
if (Exp_Year < 50)                                      b4
{
    Exp_Indicator = 2000 + Exp_Year - Today_Year;       b5
}
else
{
    Exp_Indicator = 1900+Exp_Year - Today_Year;         b6
}
...
Code Continues                                          b7
...
```

Figure 1. *Program Pseudo Code*

In this Y2K example, edge coverage, Π = { (1, 2), (1, 3), (4, 5), (4, 6) }. Two complete paths {1, 2, 4, 5} and {1, 3, 4, 6} can have already fulfill the edge coverage criteria.

Semantically, {1,2, 4, 5} tests the case:

Case 1: Both Today's Date and Expire Date in 19XX.

(1, 3, 4, 6} tests the case:

Case 2: Both Today's Date and Expire Date in 20XX.

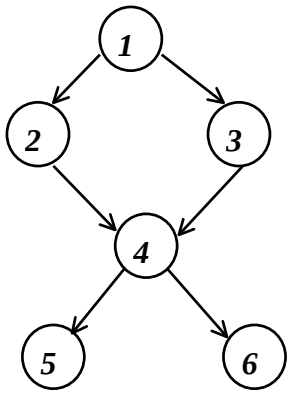

Figure 2. *Control Flow Graph*

Table 1. *Def/Use Graph*

node	c-use	def	edge	p-use
1	∅	Exp_indicator Today_Year Exp_Year	(1, 2)	Today_Year
2	Today_Year	Today_Year	(1, 3)	Today_Year
3	Today_Year	Today_Year	(4, 5)	Exp_Year
4	∅	∅	(4, 6)	Exp_Year
5	Exp_Year	Exp_Year		
6	Exp_Year	Exp_Year		
7	Today_Year Exp_year	Exp_indicator		

Unfortunately, two cross-century cases are not tested. They are:

Case 3: Today's Date in 19XX and Expire Date in 20XX.

Case 4: Today's Date in 20XX and Expire Date in 19XX.

These 2 cases should indeed be focused by the Y2K compliance test but they may be missed out by the all-edge criteria.

On the other hand, consider the all-uses (or just all-c-uses in this case) criteria, for every node i and every $x \in$ def(i), the selected paths should include a def-clear path w.r.t. x from i to all elements of dcu(x, i). Review the dcu of our example:

dcu(Today_Year, 2) = {5, 6}
dcu(Today_Year, 3) = {5, 6}

Paths from node 2 to node 5 and also node 6 are required. Furthermore, paths from node 3 to node 5 and also node 6 are required. Therefore, to satisfy the All-Uses criteria, despite paths {1, 2, 4, 5} and {1, 3, 4, 6}, {1, 2, 4, 6} and {1, 3, 4, 5} may be included. Not only the criteria c-use is stronger, but also it really reveals the semantic of the real-world testing requirements of test cases

construction. That is, all conditions are included:

Case 1: Both Today's Date and Expire Date in 19XX.

Case 2: Both Today's Date and Expire Date in 20XX.

Case 3: Today's Date in 19XX and Expire Date in 20XX.

Case 4: Today's Date in 20XX and Expire Date in 19XX.

3. Survey on Coverage Testing Tools for COBOL on Mainframe Platform

ATAC (Automatic Test Analysis for C) was developed and used as a research instrument for coverage testing in C programs [5] [6]. It has been applied [7] and commercialized as χATAC in the Software Understanding System package, χSuds. We can identify the following capabilities of ATAC in the software testing process:

- measuring test set completeness by control and data flow coverage;
- displaying non-covered code to facilitate test cases creation;
- reducing regression test set size by determining minimal test set out of total tested cases.

Potentially, ATAC can be applied to effectively select randomly generated test cases. The coverage measurement process is nearly transparent to the tester. At any time of the testing, ATAC can display summary of the coverage and uncovered codes. It can also determine minimal test set for an optimal coverage. Therefore, the selected minimal test set can be used for regression test to minimize testing cost.

We surveyed the available coverage testing tools for COBOL language on IBM mainframe platforms. Comparison was made between these coverage tools. From this survey, we found that the mainframe industry lacks the coverage tools that support the data flow coverage measure.

3.1 Status of COBOL on the Mainframe Platform

COBOL is the major high level language employed in IBM OS/390, MVS, VM mainframe environment. COBOL language is still one of the strategic and supported products of IBM mainframes. Versions of COBOL emerge continuously in the mainframe industry. Many organizations, especially the business sector, have millions of dollars invested in COBOL-based systems and in COBOL programmers who create and maintain the applications. COBOL applications are performing mission critical applications in the business world that the users do not really want to retire them.

A large number of users/programmers are very pleased with their COBOL applications, except that they simply want to move them to open systems or client/server architectures. In many cases, rewriting programs in other languages is costly and risky. Modernizing a COBOL application is often the alternative with the least cost, least developing time, and least risk. Many COBOL developers like IBM, Computer Associates (CA), ACUCORP and Intersolv have invented new versions of COBOL, modernizing COBOL compilers that supports an open system, where a client/server configuration might take relatively little time and effort to set up.

In conclusion, we believe that COBOL will last long in the mainframe platform. To renew and to import new technologies to the mainframe COBOL programming environment is necessary and rewarding.

3.2 COBOL Coverage Tools on the Mainframe

In this survey, COBOL coverage products of four major software vendors in the mainframe industry are selected for evaluation. Table 2 compares the features of these tools. *Paragraph* in COBOL is similar to *function* in C language. Paragraph coverage directs the tester to construct test cases that each paragraph in COBOL is to be covered at least once.

From Table 2, we can identify that the IBM Code Assistant possesses the most complete features. Additional features like visual aid and tracing of specified coding vary from products to products. CA-TestCoverage and IBM Code Assistant execute programs under a normal execution environment for coverage measurement while SMARTTEST and XPEDITER requires the measurement to be taken on a dedicated debugging environment.

Table 2. Mainframe COBOL Coverage Tools

Vendor	Computer Associate (CA)	VIASOFT	Compuware	IBM
Product Name	CA-TestCoverage / 2000	VIA/SmartTest with Test Coverage Analysis (TCA) option	XPEDITER/ Code Coverage	Coverage Assistant (CA)
Execution Platform	MVS, OS/390	MVS, OS/390	MVS, OS/390 and Microsoft Windows 3.X /95 (for viewing result)	MVS, OS/390
Coverage Measurement Environment	Program compiled by specific compiler and executed normally	Testing program loaded under the debug environment SmartTest	Testing program loaded under the debug environment XPEDITER	Program compiled by specific compiler and executed normally
Block Coverage Measurement	Paragraph coverage	Statement coverage, paragraph coverage	Statement coverage, paragraph coverage	Statement coverage
Control Flow Coverage Measurement	N.A	N/A	Edge coverage	Edge coverage
Data Flow Coverage Measurement	N/A	N/A	N/A	N/A
Code execution count	Present	Present	Present	N/A
Test Set Minimization	N/A	N/A	N/A	Present by complementary using Distillation Assistance (DA) under the same software package
Test cost evaluation	N/A	N/A	N/A	Execution time measured is considered as the cost
Trace changed coding	N/A	N/A	Present manually: Segments of program can be highlighted to trace	Present by complementary using of Source Audit Assistant (SAA) under the same software package
Visual aid	- Summary statistics report -Source Map Report (Indexed program source)	-Summary statistics report	-High-level, system-level graphical structural chart for IT manager -Colored code to indicate branches and complexity	-Summary statistics report -Targeted Coverage Report -Annotated Listing Report
Other features	Compare difference of execution counts for different test cases	N/A	Advise risk degree of a program based on the coverage, execution count, verb types and McCabe metric	Execution time is measured

IBM Code Assistant is able to provide advanced features like test set minimization. XPEDITER of Compuware makes use of PC platform to present the result in a graphical and user-friendly way. None of them, however, supports data flow coverage measurement.

From this product survey, we notice that the mainframe industry still lacks software testing tools to make use of the data coverage technique. In view of that, we designed and implemented a testing coverage tool named ATACOBOL for the mainframe COBOL program development.

4. Implementation

4.1 Overview

The instrumentation, coverage measurement and analysis of ATACOBOL are implemented across mainframe and PC platforms. The ATACOBOL instrumentation and analysis program tools are written in C language using the Microsoft Visual C++ 6.0 Compiler. A version of COBOL code called the S-COBOL (structured COBOL) of Application Productivity System (APS) Development Center provided by Intersolv is selected as the target language for analysis. The S-COBOL is introduced to one of the authors' working environment since 1986 and all batch programs are mostly written in S-COBOL.

4.2 Environment Setup

Coverage measurement of ATACOBOL is currently achieved across IBM OS/390 and Microsoft Windows 95 by the aid of file transfer. The steps of the whole process are illustrated in Figure 3.

4.3 ATACOBOL Architecture

The use of ATACOBOL involves 3 phases consequently:

- Instrumentation Phase: The S-COBOL source is instrumented on PC according to the structural information extracted from the source and compiled listing.
- Testing Phase: The instrumented source is compiled and testing is carried out on mainframe as usual. Program execution is traced automatically.
- Analysis Phase: The trace log is downloaded to PC and analyzed to take coverage measurement.

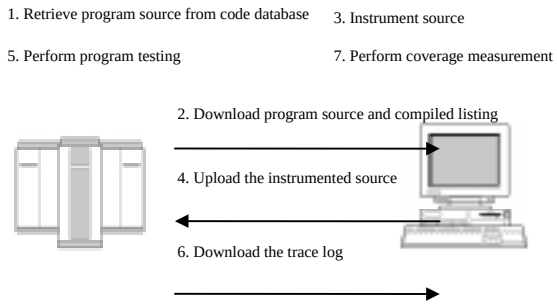

Figure 3. ATACOBOL Testing Setup

ATACOBOL is composed of four major components:

- Code Parser: The ATACOBOL Code Parser analyzes S-COBOL source code and produces two files: Control Flow Information File and Data Flow Information File. In developing the parser, some techniques used by classic compilers [8] are applied while shortcuts that take advantages from the features of S-COBOL are also considered.

S-COBOL uses indentation, not punctuation, to control program logic; therefore, blocks can be parsed easily by the indentation of statements. Control Flow Information File contains control flow information about the source program for use by the ATACOBOL Code Instrumenter. Both files are employed in the analysis phase.

To build the Data Flow Information File, macros and variables defined in the program listing are parsed to form a variable table. For each block, variables are scanned and assigned to the Def/Use graph.

- Code Instrumenter: From the Control Flow Information File, ATACOBOL Instrumenter gets information about blocks and their corresponding positions in the S-COBOL source file. Then ATACOBOL inserts codes with correct alignment to the source. The purpose of the inserted codes is to call the ATACOBOL Runtime Module passing with a unique block identifier as the parameter. This unique block identifier is composed of the paragraph number and the node number.

When a block is executed during the testing, the Runtime Module logs down the paragraph number and the block number as an identifier. As a result, the execution path can be traced.

- Runtime Trace Module: Up to the current implementation, the COBOL system call "DISPLAY" is employed as the Runtime Routine. It outputs the trace log to the SYSOUT (System Output) of OS/390 JES2 job-held queue. Its function is similar to an output file.

The SYSOUT is captured after testing as the input to ATACOBOL Coverage Analyzer. For further development, a discrete Runtime Module could be written in IBM 370 Assembly Language and writes the output to user defined trace log files. It would then be able to support specific function in the customized runtime module.

• Coverage Analyzer: The ATACOBOL Coverage Analyzer takes the Control Flow Information File, Data Flow Information File and Trace Log as inputs. It performs several levels of coverage measurements.

ATACOBOL Analyzer finally outputs reports about the coverage measurement, including a summary report of the percentage of coverage per paragraph and uncovered blocks, decision edges, c-uses or p-uses.

These components work co-operatively to perform coverage measurement as illustrated in Figure 4.

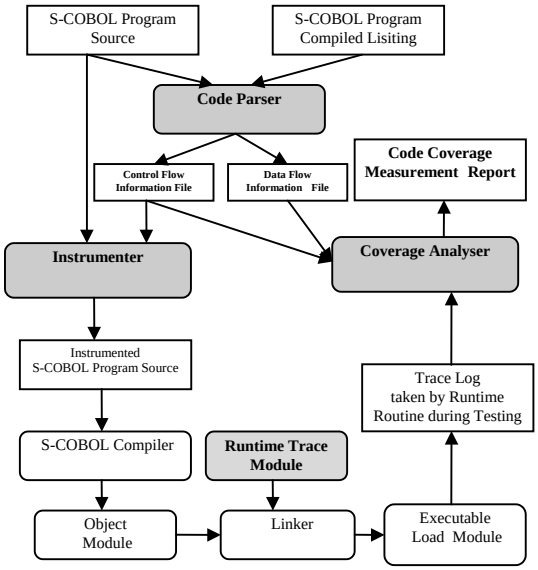

Figure 4. ATACOBOL Architecture

5. Enhanced Rules for Selecting Def/Use Pair

Modern practical computer language contains data structure of variables (e.g. *struct{}* in C language). Figure 5 shows a section of live data definition in APS COBOL. COBOL programmers used to collect variables under the same data structure label for documentation reason rather than any intrinsic relationship among the variables. Each variable is an individual counter, collected under the data (variable) labeled 'COUNTERS'.

```
05  COUNTERS.
    SKIP1
    10  LINE-CNT            PIC 9(2)  VALUE  80.
        88  PAGE-OVERFLOW               VALUE 76 THRU 80.
    10  LINE-CNT1           PIC 9(2)  VALUE  80.
        88  PAGE-OVERFLOW1              VALUE 76 THRU 80.
    10  LINE-CNT2           PIC 9(2)  VALUE  80.
        88  PAGE-OVERFLOW2              VALUE 76 THRU 80.
    10  PAGE-CNT            PIC 9(2)  VALUE  0.
    10  PAGE-CNT1           PIC 9(2)  VALUE  0.
    10  PAGE-CNT2           PIC 9(2)  VALUE  0.
```

Figure 5. Live APS COBOL Structural Data Definition

The definition of data flow coverage proposed by [2] is based on an ideal and simple language. Def/Use relationship needs to be enhanced to for adaptation to modern data structures. A formal definition of the enhanced rules are given as follows:

Let *i*, *j* be variables in a structure, and CompStruct(*i*, *j*) be a function that returns the hierarchical relationship between *i* and *j*, then

CompStruct(*i*, *j*) = TRUE
if *i* = *j*, or *i* is the parent/grandparent *j*, or *j* is the parent/grandparent *i*
CompStruct(*i*, *j*) = FALSE otherwise

Now, we re-define the two functions defined in Section 1:

- dcu(*x*, *i*) is the set of all nodes *j* such that $x \in$ c-use(*j*) and for which there is a def-clear path w.r.t. *x* from *i* to *j*, given CompStruct(*i*, *j*) = TRUE.
- dpu(*x*, *i*) is the set of all edges (*j*, *k*) such that $x \in$ p-use(*j*, *k*) and for which there is a def-clear path w.r.t. *x* from *i* to *j*, given CompStruct(*i*, *j*) = TRUE.

6. Measurement

6.1 System Description

In this section, ATACOBOL is applied with live production programs and test cases. The system under measurement is the batch programs of an interface system of a bank's mainframe application to Real Time Gross Settlement System (RTGS) in Hong Kong [9].

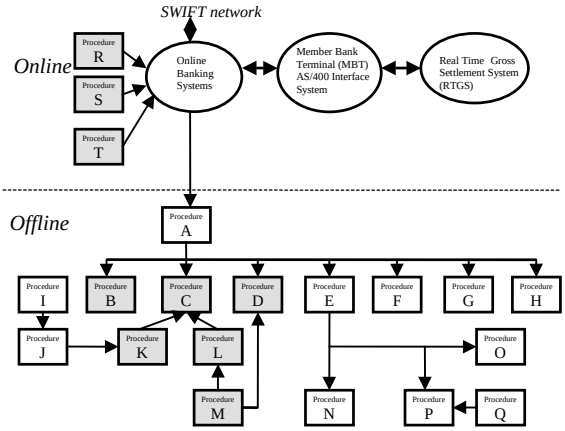

Figure 6. System Overview

Figure 6 shows the overview this system. The Online Banking Systems are developed in IBM 370 Assembly Language. A procedure consists of 1 to 5 modules, mostly written in S-COBOL language for off-line execution. There are three online procedures that create real-time spools to the Online Banking Systems.

6.2 Number of C-Use and P-Use vs. Number of Faults

Shaded procedures in figure 6 are selected for this measurement. There are totally 21 modules (say module M1 to M21) developed in APS COBOL for the selected procedures. The module history and source can be retrieved from the version control system of the development environment (see Table 3). Problem/Change Reports during March 1998 to February 1999 are also collected (see Table 4). The information is plotted in Figure 7 and Figure 8, respectively.

From the graphs, we observe two peaks in the program size changes. These two peaks, occurring in July and January, reflect two major releases at that time. Referring to Figure 8, during the first major release, the complexity of the modules (number of blocks, edges, c-uses and p-uses) increased as the number of faults reported also increased. The number of faults was reduced as program fixes were released. In September, as the number of transactions handled by the modules released in July increased, new problems broke out. This accounted for the higher fault rate in September. Overall, the graph shows that other than the number of c-uses and p-uses, other factors also affect the reliability of a software system.

However, the number of blocks, number of c-uses and number of p-uses increased are highly correlated as reflected in Figure 7. This makes it difficult to distinguish the impact of the data flow to software reliability. It requires extensive experiments to collect more statistics in order to get a clear picture on the effects of c-uses and p-uses to software reliability.

Table 3. Module Amendment Statistics

Date	Affected Modules	No. of Source Lines Increased	No. of Blocks Increased	No. of Edges Increased	No. of C-Uses Increased	No. of P-Uses Increased
Original Number	M1-M21	172,641	29,586	18,622	51,877	22,357
1998 Mar	M1-M4	1,032	28	12	36	20
1998 Apr	M5-M7	42	0	0	2	0
1998 May	M2, M13	220	6	2	21	2
1998 Jun	M11	165	4	0	9	0
1998 Jul	M2-M6, M8-M14, M18-M21	16,088	603	275	698	384
1998 Aug	M2	323	11	4	27	13
1998 Sep	M3, M8-M9, M18-M21,	647	39	5	120	10
1998 Oct	M6-M7, M14,M15	544	28	4	54	4
1998 Nov	N/A	0	0	0	0	0
1998 Dec	N/A	0	0	0	0	0
1999 Jan	M2-M6, M7-M9, M13-M17	9,556	490	182	1,103	266
1999 Feb	M1, M6-M7	461	8	0	9	0

Table 4. Module Failure Statistics

Month	No. of Problem Reported
1998 March	2
1998 April	2
1998 May	2
1998 June	1
1998 July	4
1998 August	0
1998 September	4
1998 October	1
1998 November	0
1998 December	1
1999 January	1
1999 February	1

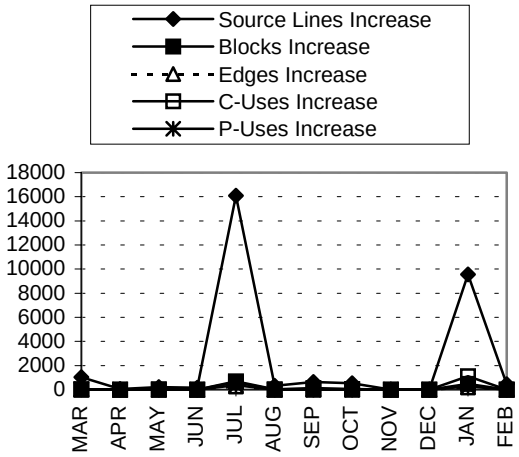

Figure 7. System Growth

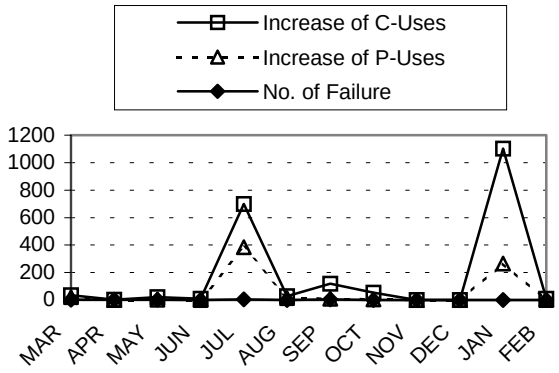

Figure 8. Number of Failure with C− and P− Uses Changes

6.3 Data Flow Coverage of Live Test Cases

Three modules (say O1, O2 and O3) are selected for coverage measurement with the system test and user acceptance test cases before their last release.

The measurement shown in Table 5 demonstrates ATACOBOL's ability to measure production scale modules. O1 is a newly created module. In the system test and user acceptance test, its functionality is tested thoroughly. On the other hand, O2 and O3 are enhanced versions. Only their enhanced features are thoroughly tested while few basic features are re-tested with representative regression test cases. The coverage for O2 and O3 is, therefore, relatively low. It would be useful if the measurement tool can focus only on the affected parts of a program enhancement.

For further measurement, we would measure the increase in number of test cases against the percentage of coverage. The behavior of growth of coverage relates to the organization of the program. If the program has evenly distributed coding on various functions, the growth curve would linearly increase. On the other hand, if the program has a large piece of common mainline, the growth curve is expected to increase fast first but gradually slow down.

Table 5. Live Test Result Using ATACOBOL

Module	O1	O2	O3
Number of Source Lines	4,276	7,225	15,044
Number of Test cases	51	12	5
Number of Blocks	831	1,473	2,206
Percentage of Block Coverage	100%	42%	18%
Number of Decision Edges	504	999	1,430
Percentage of Decision Edge Coverage	98%	33%	7%
Number of C-Uses	2,012	2,604	3,695
Percentage of C-Uses Coverage	94%	22%	5%
Number of P-Uses	838	1,317	2,121
Percentage of P-Uses Coverage	95%	28%	6%

6.4 The Usefulness of Enhanced Rules on Data Structures

In the current ATACOBOL implementation, it supports three hierarchical levels of data structure representation. Module O2 is used to compare the difference if elements of a data structure are not distinguished from each other. This experiment is achieved by modifying the variable table to wipe away level 2 and level 3 identifier of a variable.

The experimental result shown in Table 6 reveals that nearly doubled amount of c-uses and p-uses are identified by ATACOBOL if the enhanced rules are not applied. That means

almost the same amount of c-uses and p-uses are incorrectly defined and should be eliminated.

Table 6. Def/Use Pair Extracted by Different Rules

	All elements in a data structure are considered as the same	Using enhanced rules as described in section 5.
No. of c-uses found	5073	3156
No. of p-uses found	3066	1589

7. Conclusions

We have surveyed the literature about coverage techniques and evaluated practical software tools applied for coverage techniques.

ATACOBOL, a coverage measurement tool for COBOL in mainframe, is designed and implemented. ATACOBOL is written in C language. It carries out instrumentation and measurement across the mainframe and PC platforms. ATACOBOL is able to perform block, decision, c-use and p-use coverage measures.

The importance of data flow coverage criteria in identifying real-world Y2K-related problematic paths is also demonstrated. Moreover, we have enhanced the rules of data flow coverage by adapting high-level data structures for a more accurate measure.

ATACOBOL is applied to measure live programs from the banking sector with live test cases. With the extensive application of ATACOBOL, we hope to explore more about the usefulness of data flow coverage, and the relationship between coverage and reliability.

Acknowledgement

The work described in this paper was supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region (HKSAR), with Project No. CUHK4432/99E.

References

[1] Lyu, M. R. (ed.), *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.

[2] Clarke, L. A., Podgurski A., Richardson, D. J., and Zeil, S. J., "A Formal Evaluation of Data Flow Path Selection Criteria," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, 1989, pp. 1318-1332.

[3] Rapps, S., and Weyuker, E. J., "Selecting Software Test Data Using Data Flow Information," *IEEE Transactions on Software Engineering*, vol. 11, no. 4, 1985, pp. 367-375.

[4] Frankl, Phyllis G., and Weyuker, Elaine J., "An Applicable Family of Data Flow Testing Criteria," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, 1988, pp. 1483-1498.

[5] Horgan, J. R., and London, S., "A Data Flow Coverage Testing Tool for C," *Proceeding of the 2nd Symposium on Assessment of Quality Software Development Tools*, 1992, pp. 2-10.

[6] Lyu, M. R., Horgan, J. R., and London, S., "A Coverage Analysis Tool for the Effectiveness of Software Testing," *IEEE Transaction on Reliability*, vol. 43, no. 4, 1994, pp. 527-535.

[7] Horgan, J. R., London, S., and Lyu, M. R., "Achieving Software Quality with Testing Coverage Measure," *IEEE Computer*, vol. 27, no. 9, 1994, pp. 60-69.

[8] Aho, A. V., Sethi, R., and Ullman, J. D., *Compilers: Principles, Techniques, and Tools*, Addison Wesley, 1986.

[9] Beecham, B. J., *Monetary and Financial System in Hong Kong*, Hong Kong Institute of Bankers, 2nd Edition, 1998.