Logic analysis and its tools

by Dr. R. S. WANG
RCA
Moorestown, New Jersey

ABSTRACT

This article discusses the logic analysis of a program, points out problems with the logic analysis process in general use, and introduces an approach to logic analysis that is more effective and less time consuming. The new method generates logic paths out of programs and preanalyzes the paths in lieu of directly analyzing the program. Three preanalysis software tools are introduced: procedure logic path generator, program logic path generator, and logic path pre analyzer. Sample outputs are given to demonstrate the difference between program analysis and logic path analysis.
INTRODUCTION

The general process of analyzing a program consists of identifying entry point, identifying the path at branch point, going to the called routine, returning to the calling routine, manipulating data, and interpreting data. The process is completely undisciplined and is laborious and repetitive most of the time.

The logic analysis is done by different people at different stages of program development. The typical and more thorough logic analysis is done during the design stage by the designer and reviewer, and during the test stage by the tester. The efficiency and effectiveness of the logic analysis approach will cause a distinct impact on the quality and cost of the design and test.

This article suggests a systematic logic analysis approach. The approach is to isolate and preanalyze the program logic paths on the computer before further analysis. The logic analysis of a CMS-2 language program in the white box test stage is used for the illustration.

LOGIC ANALYSIS FOR MODULE VERIFICATION TEST

A system consists of a set of modules with each module consisting of a set of procedures. The test of a module is a white box test by which data presets and data outputs are needed to be defined through the logic analysis of the module and the interpretation of the test case in the test requirement. Two approaches are used to complete the module verification test (MVT): One is a machine test and the other is an inspection test.

For each test case, after the tester has acquired the understanding of the overall function to be performed, the tester begins to analyze the program and the related data design. The tester tries to correlate the test case and the program to make sure that the program can carry out the functions as described in the test case. The analysis consists of the following processes:

1. Defining data according to the preset data specified in the test case and the format required by the data design
2. Justifying and defining the preset data needed by the program yet not defined in the test case
3. Manipulating the data as instructed by the related statements in the program
4. Choosing branches at decision statements
5. Comparing the outputs of the program with the expected outputs specified in the test case

The above analysis process may iterate one or more times because of the following reasons:

1. Mistakes made in the data manipulation
2. Imperfection of the test case definition
3. Complexity of the program
4. Confirmation of the analysis result
5. Identification or confirmation of the discrepancies or errors

Problems of the Logic Analysis of MVT

The analysis is the foundation of the MVT. It is usually quite time-consuming for the tester, requiring much patience to go through the instructions primarily executed by the CPU. Figure 1 symbolically represents the nature and problems of the program analysis. Part (a) of Figure 1 represents a set of test cases. Input and output data are specified in the test case. The stars symbolize the special data format used in the test case. Part (b) represents the program where many logic paths are blended together in a box. The data, which are the inputs and outputs of the logic paths, are expressed in definite formats as symbolized by the circles. Each test case in Part (a) will have a corresponding logic path in part (b).

A problem of the prevalent logic analysis is that, for each analysis, one has to identify the tangled logic path of the program while performing other analysis efforts. This impacts the required time and quality of the total analysis.

For short and straightforward logic paths, the problem is not severe. However, for lengthy and tangled logic paths, the problem is severe. For each analysis one has to memorize the test conditions, select branch at decision statements, manipu-
late data, flip the pages to locate the called procedures and calling procedures, and memorize control indicators. Because of the involvements, one may easily lose track and end up repeating the analysis or obtain analysis of questionable accuracy. The uncertainty may appear in the result of machine test. It may result in another cycle of analysis and machine test. For inspection test, this simply means a questionable test quality. One of the causes of the problem is directly analyzing the program each time.

In doing the analysis, visibility is a critical factor that affects the analysis effectiveness. Because the program contains many logic paths not related to the test case, the related logic path scatters around in the program, and the visibility is greatly impacted.

LOGIC PATH ISOLATION

To increase the visibility, to save time, and to improve the analysis effectiveness, it is helpful if logic paths can be isolated from the program for further analysis. Logic path isolation is symbolically depicted in Figure 2. The program shown in Figure 2(b) is transformed into a set of isolated logic paths shown in Figure 2(c). For further analysis, one can check the test case against the isolated logic path instead of checking the test case against the whole program.

The decision statements in the path become condition descriptors. They reflect the preset conditions, data, and the derivatives of the preset conditions and data. After the logic path is isolated, to analyze a logic path one does not have to flip the pages back and forth to locate a called procedure or a calling procedure. One does not have to select a branch at every decision statement either. The analysis of the logic path is simplified and straightforward. The tester can make a direct comparison between the conditions indicated in the decision statements with the data presets in the test case. The expected outputs of the test case can be compared directly with the outputs of the imperative statements. The redundancy of the logic path identification effort is eliminated. The visibility of the logic is greatly increased. Software tools to carry out the logic path isolation processes are discussed in the following paragraphs.

Procedure Logic Path Generator

This generator uses the program as input. It converts each procedure into a set of logic paths. Each logic path is identified. Files are created. Printed output is also available. An example is shown in Figure 3 to illustrate the process. A software system has one module which consists of two procedures, PROC1 and PROC2, as shown in Figure 3(a) and 3(b). By using the program as input, the generator produces one set of three logic paths for PROC1 as shown in Figure 3(c), and one set of three logic paths for PROC2 as shown in Figure 3(d). The decision statements are marked so that they can be analyzed separately and do not interfere with the imperative statements. The decision statements in the program become condition descriptors in the generated logic path. The END statements of the program are not shown in the logic path. Since all the unrelated information is not shown in the generated logic path, obviously the visibility is significantly increased. For each logic path, analyzing the generated logic path is much easier and more effective than analyzing the logic path in the program procedure.

Module Logic Path Generator

After the logic path has been identified for a test case a sequence of procedure logic path IDs are manually listed. This sequence of procedure logic path IDs is entered as the input to the module logic path generator. The generator then accesses the procedure logic path files for the selected IDs. The procedure calls can be resolved automatically during the generation process. The module logic paths are shown in Figure 4. Like the procedure logic paths, the module logic paths present a higher visibility for logic analysis than the logic paths in the program procedure.

Logic Path Preanalyzer

The outputs of the procedure logic path generator and the outputs of the module logic path generator can be further organized by a tool called a logic path preanalyzer. The pur-
Logic Analysis and Its Tools

Figure 3—(a) Flowchart of PROCEDURE PROC1; (b) Flowchart of PROCEDURE PROC2; (c) Procedure logic paths of PROC1; (d) Procedure logic paths of PROC2
The analysis of the logic path is straightforward. Using logic path 1 as an example, it shows that if the conditions $D_1$ and $D_2$ are 0, the $A$, $C$, $E$, and $F$ will be set to 1. $B$ will be set to $X$. $X$, $D_1$, and $D_2$ are the inputs needed by the logic path. The input and output data provide critical information for module interface analysis if more than one module exists in a system.

For logic path 2 the tester can easily tell that the second and third conditions are contradictory, thus, the logic path is considered invalid. The visibility of the generated logic paths is clearly much better than that of the program. Redundant page flipping and branch selection are eliminated from the analysis effort.

CONCLUSION

The approach introduced is used to isolate a logic path in a program for an MVT test case. The immediate purpose is to increase the visibility of the related program logic to be analyzed. Judging by the author’s experience, the approach may improve the test quality and reduce the test cost. Moreover, the logic paths generated may constitute part of the MVT test results. They can be used for reviews and the analysis of integration test. The integration test is the next level of test after MVT.

Since the basic concept of this approach is to help analyze programs effectively and economically, the applicability of this approach is not limited in MVT. It can be applied to the procedure test and the integration test. It is even applicable in the design state and the maintenance stage. Since the logic path is considered invalid. The visibility of the generated logic paths is clearly much better than that of the program. Redundant page flipping and branch selection are eliminated from the analysis effort.

CONCLUSION

The approach introduced is used to isolate a logic path in a program for an MVT test case. The immediate purpose is to increase the visibility of the related program logic to be analyzed. Judging by the author’s experience, the approach may improve the test quality and reduce the test cost. Moreover, the logic paths generated may constitute part of the MVT test results. They can be used for reviews and the analysis of integration test. The integration test is the next level of test after MVT.

Since the basic concept of this approach is to help analyze programs effectively and economically, the applicability of this approach is not limited in MVT. It can be applied to the procedure test and the integration test. It is even applicable in the design state and the maintenance stage. Since the logic path is considered invalid. The visibility of the generated logic paths is clearly much better than that of the program. Redundant page flipping and branch selection are eliminated from the analysis effort.

CONCLUSION

The approach introduced is used to isolate a logic path in a program for an MVT test case. The immediate purpose is to increase the visibility of the related program logic to be analyzed. Judging by the author’s experience, the approach may improve the test quality and reduce the test cost. Moreover, the logic paths generated may constitute part of the MVT test results. They can be used for reviews and the analysis of integration test. The integration test is the next level of test after MVT.

Since the basic concept of this approach is to help analyze programs effectively and economically, the applicability of this approach is not limited in MVT. It can be applied to the procedure test and the integration test. It is even applicable in the design state and the maintenance stage. Since the logic path is considered invalid. The visibility of the generated logic paths is clearly much better than that of the program. Redundant page flipping and branch selection are eliminated from the analysis effort.

CONCLUSION

The approach introduced is used to isolate a logic path in a program for an MVT test case. The immediate purpose is to increase the visibility of the related program logic to be analyzed. Judging by the author's experience, the approach may improve the test quality and reduce the test cost. Moreover, the logic paths generated may constitute part of the MVT test results. They can be used for reviews and the analysis of integration test. The integration test is the next level of test after MVT.

Since the basic concept of this approach is to help analyze programs effectively and economically, the applicability of this approach is not limited in MVT. It can be applied to the procedure test and the integration test. It is even applicable in the design state and the maintenance stage. Since the logic path is considered invalid. The visibility of the generated logic paths is clearly much better than that of the program. Redundant page flipping and branch selection are eliminated from the analysis effort.

CONCLUSION

The approach introduced is used to isolate a logic path in a program for an MVT test case. The immediate purpose is to increase the visibility of the related program logic to be analyzed. Judging by the author's experience, the approach may improve the test quality and reduce the test cost. Moreover, the logic paths generated may constitute part of the MVT test results. They can be used for reviews and the analysis of integration test. The integration test is the next level of test after MVT.

Since the basic concept of this approach is to help analyze programs effectively and economically, the applicability of this approach is not limited in MVT. It can be applied to the procedure test and the integration test. It is even applicable in the design state and the maintenance stage. Since the logic path is considered invalid. The visibility of the generated logic paths is clearly much better than that of the program. Redundant page flipping and branch selection are eliminated from the analysis effort.

CONCLUSION

The approach introduced is used to isolate a logic path in a program for an MVT test case. The immediate purpose is to increase the visibility of the related program logic to be analyzed. Judging by the author's experience, the approach may improve the test quality and reduce the test cost. Moreover, the logic paths generated may constitute part of the MVT test results. They can be used for reviews and the analysis of integration test. The integration test is the next level of test after MVT.

Since the basic concept of this approach is to help analyze programs effectively and economically, the applicability of this approach is not limited in MVT. It can be applied to the procedure test and the integration test. It is even applicable in the design state and the maintenance stage. Since the logic path is considered invalid. The visibility of the generated logic paths is clearly much better than that of the program. Redundant page flipping and branch selection are eliminated from the analysis effort.

CONCLUSION

The approach introduced is used to isolate a logic path in a program for an MVT test case. The immediate purpose is to increase the visibility of the related program logic to be analyzed. Judging by the author's experience, the approach may improve the test quality and reduce the test cost. Moreover, the logic paths generated may constitute part of the MVT test results. They can be used for reviews and the analysis of integration test. The integration test is the next level of test after MVT.

Since the basic concept of this approach is to help analyze programs effectively and economically, the applicability of this approach is not limited in MVT. It can be applied to the procedure test and the integration test. It is even applicable in the design state and the maintenance stage. Since the logic path is considered invalid. The visibility of the generated logic paths is clearly much better than that of the program. Redundant page flipping and branch selection are eliminated from the analysis effort.

CONCLUSION

The approach introduced is used to isolate a logic path in a program for an MVT test case. The immediate purpose is to increase the visibility of the related program logic to be analyzed. Judging by the author's experience, the approach may improve the test quality and reduce the test cost. Moreover, the logic paths generated may constitute part of the MVT test results. They can be used for reviews and the analysis of integration test. The integration test is the next level of test after MVT.

Since the basic concept of this approach is to help analyze programs effectively and economically, the applicability of this approach is not limited in MVT. It can be applied to the procedure test and the integration test. It is even applicable in the design state and the maintenance stage. Since the logic path is considered invalid. The visibility of the generated logic paths is clearly much better than that of the program. Redundant page flipping and branch selection are eliminated from the analysis effort.

CONCLUSION
analysis is the major effort in software development and maintenance, the approach deserves further discussion.

The design, design review, procedure test, module verification test, and integration test are consecutive processes in the program development cycle. Similar logic path analyses are done in each of the processes as was done for the module verification test. In fact, they all present more or less the same problems as mentioned above. The logic path isolation and analysis efforts that may be useful to the next process are not saved. Consequently one may have to repeat the work. This is a waste. For example, the designers must check the logic paths in the flowchart to compare with the description in the specification. The design reviewers also have to do the same. If the generated logic path analysis approach is used, the designer may generate the logic paths not only for his own use, but it will also be helpful to design reviewers who analyze the design. Furthermore, logic paths generated may be more suitable for presentation during the review meeting. Likewise, in addition to the usage in that test process, the procedure logic paths generated in the procedure test process may be used directly in the module verification test process.

For the MVT process, the module logic paths generated are extremely useful for the modular interface analysis of the integration test. Because of their higher readability, the machine outputs of the logic path generators and preanalyzer provide a good medium for reviews, presentations, and reports. As the validation process is pushed from the test stage into the design stage, the need for a good medium in the design stage becomes apparent.

In the test stage and the maintenance stage the program is analyzed and the tools are applicable. In order to apply the approach and develop tools for the design usage, it is recommended that a program design language be used in the design process.

In industry, long years of software development experience have demonstrated that people are not satisfied with an ad hoc development approach even though the ad hoc approach may allow more freedom and demand fewer tools. The additional freedom may allow people to see some products earlier. Fewer tools may give a feeling of saving some development cost. Now the state of the art is to impose more control and use more software tools. Generally, for large program development, better control enables analysts to see the final product earlier and the software tools can lower the total development cost.

REFERENCES
MANAGEMENT

This track addresses specific concerns in information systems staffing, management, and planning. The subtrack on staffing includes sessions on the role of women in systems, software project management, and improving staff effectiveness and productivity. The second subtrack, on software maintenance, has sessions dealing with the management of software maintenance, applications of software engineering, technical issues in maintenance, and motivation of the software maintenance programmer. The final subtrack is on planning and control; it comprises a session on planning and one on the audit of complex computer/communication systems. All told, more than 50 speakers, with experience as developers, implementers, users, and managers of computer technology, will present papers or serve on panels in this track.

EDUCATION

In a relatively short time the computer has had a profound impact on educational processes throughout our society. Computer literacy is becoming necessary for effective functioning in an increasingly complex environment. This is especially true for people working in organizations. Computer literacy is discussed in terms of differences between what industry expects of a computer-literate employee and what higher education plans to produce. Another session deals with creative uses of educational technology. State-of-the-art uses of videodiscs in industry, schools, and universities will be discussed.