Research on Software Test Coverage Analysis Methods Under DO-178C

Shuang Chen

Shanghai Aircraft design and research institute, Shanghai, China

Chenshuang@comac.cc

Abstract. Based on the DO-178C's software test coverage analysis objective requirements, combined with the actual situation of software engineering, three types of test coverage analysis methods are sorted out and studied. The method is applicable to civil aircraft airborne software testing activities, and also applicable as a kind of supplementary analysis for general software testing.

Keywords: Airborne Software; DO-178C; Airworthiness; MC/DC; Data Coupling; Control Coupling

1 Introduction

In the field of civil aircraft airworthiness, DO-178C[1] puts forward the requirements of software test coverage analysis in the process of civil aircraft development life cycle, with the main purpose of evaluating the adequacy and completeness of software test work. Based on the basic requirements of test coverage analysis proposed by DO-178C, this paper proposes strategies and methods to meet the requirements of airworthiness test coverage analysis in combination with the actual situation of software engineering.

2 Test Coverage Analysis Requirements

The requirements for test coverage analysis are presented in Objectives 3 through 8 in Appendix Table A-7 Validation of Validation Process Results of DO-178C, as shown in Figure 1.

C. Chen et al. (eds.), *Proceedings of the 3rd International Conference on Digital Economy and Computer Application (DECA 2023)*, Atlantis Highlights in Computer Sciences 17,
https://doi.org/10.2991/978-94-6463-304-7_29

Table A-7 Verification of Verification Process Results

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Test procedures are correct.	6.4.5.b	6.4.5	●	○	○		Software Verification Results	11.14	②	②	②	
2	Test results are correct and discrepancies explained.	6.4.5.c	6.4.5	●	○	○		Software Verification Results	11.14	②	②	②	
3	Test coverage of high-level requirements is achieved.	6.4.4.a	6.4.4.1	●	○	○	○	Software Verification Results	11.14	②	②	②	②
4	Test coverage of low-level requirements is achieved.	6.4.4.b	6.4.4.1	●	○	○		Software Verification Results	11.14	②	②	②	
5	Test coverage of software structure (modified condition/decision coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●				Software Verification Results	11.14	②			
6	Test coverage of software structure (decision coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●	●			Software Verification Results	11.14	②	②		
7	Test coverage of software structure (statement coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●	●	○		Software Verification Results	11.14	②	②	②	
8	Test coverage of software structure (data coupling and control coupling) is achieved.	6.4.4.d	6.4.4.2.c 6.4.4.2.d 6.4.4.3	●	●	○		Software Verification Results	11.14	②	②	②	
9	Verification of additional code, that cannot be traced to Source Code, is achieved.	6.4.4.c	6.4.4.2.b	●				Software Verification Results	11.14	②			

Fig. 1. DO-178C Test Coverage Analysis Objectives.

The test coverage analysis required by DO-178C can be broken down into three categories:

- Requirements coverage analysis;
- Structural coverage analysis;
- Data coupling and control coupling analysis.

In this paper, we will explain each of the above three analysis methods in terms of definition, scope of application, and analysis method.

2.1 Requirements coverage analysis

Requirements coverage analysis is the process of analysing the correspondence between test cases and the software high-level and low-level requirements to determine whether the selected test cases can satisfy:

- Each requirement has corresponding test cases which cover both normal functionality tests and robustness tests;
- Each test case can be traced to the corresponding requirement.

In response to the above requirements coverage analysis criteria, the coverage analysis for high-level requirements applies to DO-178C-defined Level A, B, C, and D

software, and the coverage analysis for low-level requirements applies to DO-178C-defined Level A, B, and C software.

2.2 Structural coverage analysis

Structural coverage analysis reflects how well the software source code implements the software requirements and is intended to demonstrate the adequacy of requirements-based testing by providing evidence that the structure of the code has been verified to the level of rigor required by the software level.

The structural coverage criteria required by DO-178C include:

- Statement Coverage (SC);
- Decision Coverage (DC);
- Modified Condition/Decision Coverage (MC/DC).

In this paper, we will illustrate the basic ideas and coverage methods of the 3 types of structural coverage with the example in Figure 2.

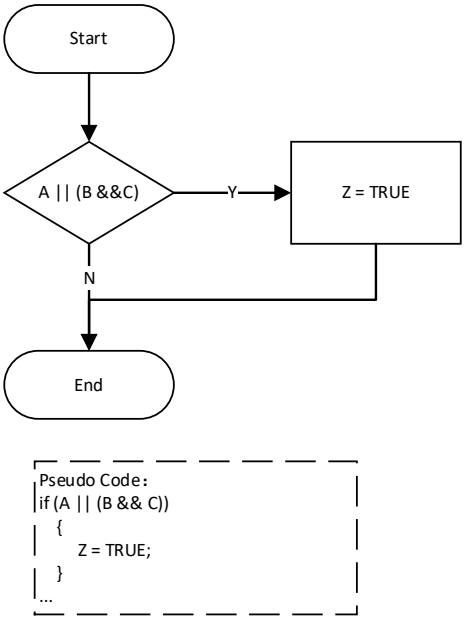

Fig. 2. Example of structural coverage.

As seen in Figure 2, the example contains 2 statements, 1 decision, and 3 conditions:

- Statement 1: if (A || (B && C));
- Statement 2: Z = TRUE;
- Decision 1: (A || (B && C);
    - Condition 1 : A;
    - Condition 2 : B;
    - Condition 3 : C.

2.2.1 Statement coverage (SC)

The statement coverage objectives apply to Level A, B, and C software as defined in DO-178C. The relevant definitions used in this paper are listed below:

- Statement Coverage: Every statement in the program has been invoked at least once.

Taking the program in Figure 2 as an example, to execute each statement in the program at least once, i.e., to satisfy the SC coverage requirement, only one test case TC-1 is needed, as shown in Table 1.

Table 1. Test cases that satisfy statement coverage

Test case	Input A	Input B	Input C	Decision Result	Output Z
TC-1	T	T	F	T	T

In the above example, although it is possible to cover all statements, the test design does not consider all cases of the value of **Decision 1**, nor does it focus on the value of each condition in **Decision 1**. Therefore, we consider statement coverage to be a weaker structural coverage criteria.

2.2.2 Decision coverage (DC)

The decision coverage objectives apply to Level A and B software as defined in DO-178C. The relevant definitions used in this paper are listed below:

- Decision Coverage: Every point of entry and exit in the program has been invoked at least once and every decision in the program has taken on all possible outcomes at least once.
- Condition: A Boolean expression containing no Boolean operators except for the unary operator (NOT).
- Decision: A Boolean expression composed of conditions and zero or more Boolean operators. If a condition appears more than once in a decision, each occurrence is a unique condition.

Take C language for example, the statements in a program that contain decisions include:

- Double-valued decision:
  - if...else if...else;
  - while...;
  - do...while...;
  - for;
- Multivalued decision:
  - Switch...case...;

The basic idea of decision coverage and coverage methods are illustrated below by the example in Figure 2.

As seen in Figure 2, two test cases TC-2 and TC-3 (see Table 2) are needed for each exit and entry in the program to be executed at least once and for all possible outcomes of each decision to be valued at least once, i.e., to satisfy the decision coverage criteria.

Table 2. Test cases that satisfy statement coverage

Test case	Input A	Input B	Input C	Decision Result	Output Z
TC-2	T	F	F	T	T
TC-3	F	F	F	F	X[a]

[a] X means that the value of Z is unknown, because the value of Z was not explicitly changed when the decision is determined to be false in this example.

It can be seen that TC-2 and TC-3 consider all possible values of **Decision 1** and are also able to cover all values of **Condition A**, but still do not consider all values of B and C and their effect on the outcome of **Decision 1**. This introduces a more comprehensive coverage as MC/DC criteria.

2.2.3 Modified condition/decision coverage (MC/DC)

Modified condition/decision coverage objectives apply to Level A software as defined in DO-178C. The relevant definitions used in this document are listed below:

- Modified Condition/Decision Coverage: Every entry and exit point in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome. A condition is shown to independently affect a decision outcome by: (1) varying just that condition while holding fixed all other possible conditions, or (2) varying just that condition while holding fixed all other possible conditions that could affect the outcome.

The modified condition/decision coverage, which is rarely mentioned in other software engineering fields, was developed specifically for the aviation industry and is intended to serve as a comprehensive criteria for evaluating software test completion with more comprehensive coverage.

The basic idea of modified condition/decision coverage and the coverage method are illustrated below with the example in Figure 2.

As seen in Figure 2, for the test to satisfy the modified condition/decision coverage criteria, it is necessary to consider the two values of T/F for **Decision 1**, as well as the respective values of T/F for **Conditions A**, **Conditions B**, and **Conditions C**, and that these conditions independently affect the decision outcome.

By definition, at least N+1 test cases (N is the number of conditions determined) are needed to fulfill the above criteria. In this case N is 3, so one possible combination of test cases contains 4 TCs, as shown in Table 3.

Table 3. Test cases that satisfy modified condition/decision coverage

Test case	Input A	Input B	Input C	Decision Result	Output Z
TC-4	T	T	F	T	T
TC-5	F	T	F	F	X[a]

TC-6	F	T	T	T	T
TC-7	F	F	T	F	X[a]

[a] X means that the value of Z is unknown, because the value of Z was not explicitly changed when decision is determined to be false in this example.

TC-4 and TC-5, which together satisfy **Condition A** independently affect the decision results; TC-6 and TC-7, which together satisfy **Condition B** independently affect the decision results; and TC-5 and TC-6, which together satisfy **Condition C** independently affect the decision results.

Compared to SC and DC, we consider MC/DC to be a more complete structural coverage criteria This is because it focuses not only on all the cases of all the values of all the decisions in the program, but also on all the possible executions of each condition in the decisions.

2.3 Data coupling and control coupling analysis

The data coupling and control coupling analysis objectives apply to Level A, B, and C software as defined by DO-178C. The relevant definitions used in this paper are listed below:

- Data Coupling: The dependence of a software component on data not exclusively under the control of that software component.
- Control Coupling: The manner or degree by which one software component influences the execution of another software component.
- Software Component: A self-contained software module/unit/function that performs a clear function of the program.

Data coupling and control coupling analysis actually belongs to one aspect of structural coverage analysis, and like SC, DC, and MC/DC analysis, data coupling and control coupling analysis is performed for requirements-based software testing and is a measure of the adequacy of software integration testing and/or hardware and software integration testing. Its purpose is to ensure that the interactions and dependencies between individual software modules/components are correct and that the components interact with each other as intended by the software design.

Common data coupling scenarios may include component A using values calculated by component B, or using global variables, or component A exchanging input and output information with component B by data parameter passing.

Common control coupling scenarios may include component A controlling a functional branch of component B by passing parameters.

Data coupling and control coupling are concerned with the actual dependencies between software components, and the article by Chilenski and Kurtz[2] identifies four types of coupled dependencies:

- Sequencing dependencies, a part of control coupling, are requirements on the execution order of modules, components, and applications.
- Timing dependencies, a part of control coupling, are requirements on the timing of individual modules, components, applications, and sequences of multiple components.

- Control flow dependencies, part of control coupling, are represented by control dependences between modules, components, and applications. This is divided into sequencing dependencies and data dependencies within branch points.
- Information flow dependencies, part of data coupling, are represented by data flows between modules, components, and applications where one module, component, or application defines the value of an object/data item that is used in another module, component, or application (data dependences).

The call tree, the compiler's linking instructions, and link mapping are a few mechanisms that affect these dependencies. These mechanisms depend on the programming language, runtime support, and hardware used. Study of the above is beyond the scope of this paper.

3 Test coverage analysis methods

In conjunction with the definitions of the three test coverage analysis methods in the previous section, this paper presents here the recommended specific implementation of coverage analysis.

3.1 Requirements coverage analysis methods

The activity of requirements coverage analysis can be performed after the software test cases and protocols have been prepared. Requirements coverage analysis can usually only be accomplished by manual analysis, and the analysis procedures described in this paper are as follows:

- The software requirements are used as an index to find out all the test cases corresponding to a software requirement with the help of a traceability matrix between software requirements and software test cases.
- Analyse the test points present in the software requirements, which may include:
    - Normal function point of the software.
    - Branches.
    - Normal value range (equivalence class);
    - Abnormal value range.
    - Boundary values.
    - Time-dependent multi cycle.
    - Trigger conditions and state migration for state machines.
    - Invalid transitions of state machines.
    - Software robustness considerations.
    - Other special case for software testing design.
- Ensure that all test points are covered by at least one test case.
- Add test cases for uncovered test points.
- For the content that is not clear in the requirements, such as the behaviour or handling of the abnormal state of the software, etc., the test cases should be supplemented first after adding additional information in the requirements.

Requirements coverage analysis is considered not complicated, and the above steps should be completed separately for test coverage analysis of high-level requirements and test coverage analysis of low-level requirements.

3.2 Structural coverage analysis methods

Structural coverage analysis is usually analysed by tools due to the huge amount of work and well-defined rules. Currently, there are a large number of mature automated tools supporting SC, DC, MC/DC analysis, such as Testbed, C++ Test, RTRT, Code Test, Vector Cast, Cantata, Tessy, Test Grid, etc.

The work procedures for conducting a structural coverage analysis using the tool include:

- Deployment of tool environments, generally supported by the tool developer.
- Write requirements-based test cases and test procedures.
- Use tools to perform automated insertion of stakes, staking and generation of test drivers.
- Execute automated tests using a tool (some tools also support tests that do not require the tool to be used to execute them) on instrument code.
- The tool analyses the coverage raw data generated after the instrument code has passed the test execution and generates a report on the structural coverage results corresponding to this test execution.
- Manually review the structural coverage results report to find the code that indicates that it is not covered and provide a reasonable explanation.

Since the above methods trusts the coverage information collected by the tool, tool qualification is needs for the structural coverage analysis tool in accordance with DO-330[3] as required by DO-178C.

Generally, the coverage analysis report generated by the tool may detect some uncovered code structures that require additional manual analysis, typically including:

- Protective code, such as extra default statements in case statements, unreachable else branches in if statements, array bounds checking, data range checking, and so on.
- Deactivated code, including code that is not intended to be executed in any configuration of the product and code that can only be executed in certain configurations of the target environment.
- Exception handling code, which can only be executed to in some special cases and cannot be executed by testing.

There are also a lot of research being done on MC/DC compliant test methods. Rohaida Romli[4] proposed a method that utilizes MC/DC coverage criteria to support more thorough automated test data generation for dynamic-structural testing in automatic programming assessment. LIU Huiying[5] proposed an improved Whale Genetic Algorithm for generating test data required for unit testing MC/DC coverage. Hong WJ[6] conducted a number of experiments using advanced symbolic execution tools to study the impact of compiler optimization on the use of symbolic execution to satisfy program MC/DC.

3.3 Methods for analyzing data coupling and control coupling

The pre-work of data coupling and control coupling analysis should be initiated in the software design phase. This paper presents a proposed methodology for data coupling and control coupling analysis, with specific steps including:

- Review software architecture design: Confirms that data structures, software architecture, internal and external inputs/outputs, data and control flows, scheduling processes and inter-processor/inter-task communication mechanisms, partitioning methods, and descriptions of software components are defined in the architectural design.
- Review software source code: Confirm that the source code conforms to the data and control flows in the software architecture.
- Analyse software architecture: Identify the specific coupling relationships in each data coupling and control coupling in the architectural design and determine the corresponding coupling coverage criteria and coupling coverage points.
- Analyse the requirements-based test cases and select the test cases related to the coupling points.
- Execute tests, record test results, and analyse coverage of data coupling and control coupling during testing.
- For coupling points that fail the test, or lack test cases, analyse the reason, supplement the test cases if necessary, and correct them.
- Summarize the results of the data coupling and control coupling analyses and prepare a coupling analysis report.

In the coupling analysis report, the sources of coupling are detailed in a paired consumer and producer format. For each data coupling, the component that produces the data and the component that consumes the data are covered in at least one test case. For each control coupling, the caller component and the called component are verified in at least one test case. To visualize and understand this information more visually, use a spreadsheet that lists each pair of couplings and the corresponding validation cases.

There have also been many other studies on data coupling and control coupling in recent years. Xia XF[8] used the radio interface software as an example, the methodology and process for meeting the data coupling and control coupling coverage objectives of DO-178C are described in detail. Kong DQ[9] summarized the issues related to data coupling and control coupling that should be considered in all aspects of software planning, development, and validation

4 Experiments and Analysis

We compare 3 types of structural coverage methods in a real C program called TCAS which includes branch such as if…else, switch, while, and multi-conditions decisions. All resources are obtained from SIR (Software Infrastructure Repository) [10].

The requirement specification of TCAS indicates that it has 12 input variables including 10 Integer and 2 Boolean variables. The detail information of these inputs and the required binary bits for conversion are shown in Table 4.

Table 4. Input Variables Information for TCAS

Variable Name	Type	Value
Cur_Vertical_Sep	int	[0,2000]
High_Confidence	bool	0,1
Two_of_Three_Reports_Valid	bool	0,1
Own_Tracked_Alt	int	[0,6000]
Own_Tracked_Alt_Rate	int	[0,1000]
Other_Tracked_Alt	int	[0,6000]
Alt_Layer_Value	int	0,1,2,3
Up_Separation	int	[0,1000]
Down_Separation	int	[0,1000]
Other_RAC	int	0,1,2
Other_Capability	int	1,2
Climb_Inhibit	int	0,1

We generated 1000 test random test cases and executed them on TCAS and then collected coverage information. Table 5 and Figure 3 show the coverage curves of the experiment results on the TCAS program.

Table 5. Coverage for SC, DC and MC/DC

Num of Test Cases	SC	DC	MC/DC
50	57.35%	12.50%	24.32%
100	91.18%	62.50%	43.24%
150	91.18%	62.50%	43.24%
200	91.18%	62.50%	54.05%
250	91.18%	62.50%	54.05%
300	91.18%	62.50%	54.05%
350	91.18%	62.50%	54.05%
400	91.18%	62.50%	54.05%
450	91.18%	62.50%	54.05%
500	94.12%	75.00%	59.46%
550	95.59%	81.25%	59.46%
600	95.59%	81.25%	59.46%
650	95.59%	81.25%	59.46%
700	95.59%	81.25%	59.46%
750	95.59%	81.25%	64.86%
800	95.59%	81.25%	64.86%
850	97.06%	87.50%	76.58%
900	97.06%	87.50%	76.58%
950	98.53%	93.75%	83.78%
1000	98.53%	93.75%	83.78%

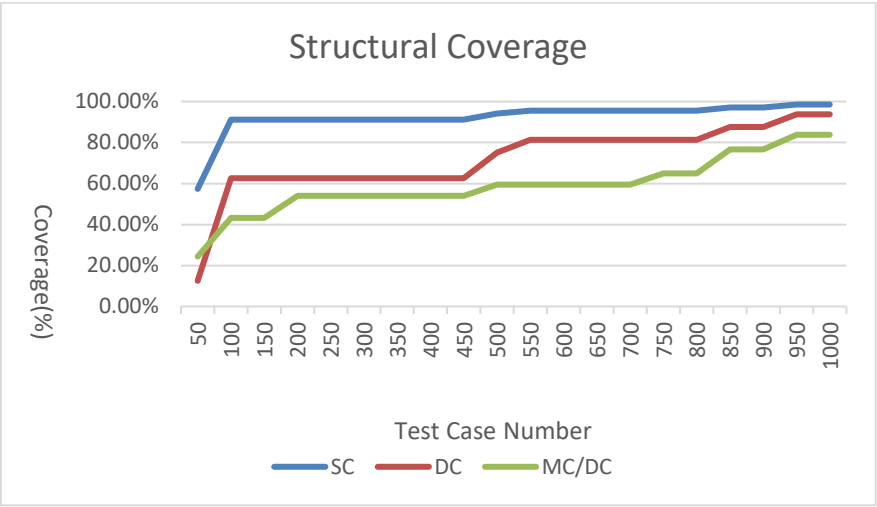

Fig. 3. Structural Coverage for TCAS Program

As mentioned earlier, 1000 random test cases were used in this experiment. We did not intentionally supplement or modify the test cases, so all types of coverage fell short of 100%. In particular, it should be noted that there is unreachable code structure in the TCAS program, so that even the SC does not reach 100%.

From this experiment, SC is the easiest of the three structural coverings to satisfy, while MC/DC is the most difficult. Therefore, it is appropriate and reasonable for DO-178C to impose structural coverage requirements corresponding to different levels of airborne software.

5 Conclusion

This paper examines and summarizes the requirements and analysis methods for test coverage analysis of airborne software. Further research includes practicing and improving the methodology in more actual airborne software projects.

References

1. RTCA. (2011) DO-178C Software Considerations in Airborne Systems and Equipment Certification. Washington.
2. Chilenski J and Kurtz J. (2007) Object-Oriented Technology Verification Phase 2 Handbook—Data Coupling and Control Coupling. DOT/FAA/AR-07/19: 12-13.
3. RTCA. (2011) DO-330 *Software Tool Qualification Considerations*. Washington.
4. Rohaida R, Shahadath S, Mazni O, Musyrifah M. (2020) Automated Test Cases and Test Data Generation for Dynamic Structural Testing in Automatic Programming Assessment

Using MC/DC. International Journal on Advanced Science, Engineering and Information Technology. Volume 10 , Issue 1. 120-127.
5. Liu HY et al. (2022) MC/DC Test Data Generation Algorithm Based on Whale Genetic Algorithm. Instrumentation,9(02):1-12.
6. Hong WJ et al. (2020) Symbolic execution compilation optimization for MC/DC. Frontiers of Information Technology & Electronic Engineering,21(09):1267-1285.
7. Zhu WZ. (2022) Coverage analysis work for data coupling and control coupling. China Science and Technology Information,2022(16):28-30.
8. Xia XF. (2019) Realization of data coupling and control coupling coverage objectives for radio interface unit software. Computer Applications and Software,36(01):34-38+44.
9. Kong DQ. (2018) An overview of the realization of data coupling and control coupling objectives in DO-178C. Aviation Computing Technology,48(05):56-58.
10. D. Hyunsook, E. Sebastian, R. Gregg (2005), Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact, Empirical Software Engineering, 10(4): 405-435.