

Certification Authorities Software Team (CAST)

Position Paper CAST-17

Structural Coverage of Object Code

Completed June 2003

(Rev 3)

NOTE: This position paper has been coordinated among the software specialists of certification authorities from the United States, Europe, and Canada. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.

Structural Coverage of Object Code

1.0 Purpose

The purpose of this paper is to present the certification authorities' concerns and position regarding the analysis of structural coverage at the object code level rather than at the source code level, particularly for Level A software. This coverage approach has been proposed by some applicants and developers for Level A software as an equivalent to replace the traditional modified condition/decision coverage (MC/DC) typically performed at the source code level, and proposes to preclude the necessity to perform source code to object code traceability (needed only for Level A software). The topic of source code to object code traceability is addressed in CAST-12 paper, entitled *Guidelines for Approving Source Code to Object Code Traceability* [1]. Some applicants have also proposed object code level structural coverage for Level B software components (needing decision coverage and statement coverage) and Level C software components (needing statement coverage); and as a supplemental method to satisfying the data coupling analysis and control coupling analysis structural coverage objective (DO-178B/ED-12B [2], section 6.4.4.2.c. and Appendix A, Table A-7, Objective 8). This paper focuses on the proposals for Level A applications; however, much of the discussion may also be applicable to lower level applications as well.

Note: For purposes of this paper, object code is typically assembly language-like instructions – this paper is not applicable to approaches that propose to provide equivalent structural coverage at the machine language level.

2.0 Background

In recent times, several applicants have proposed meeting Objective #5 of DO-178B/ED-12B [2] Table A-7 (MC/DC) by performing structural coverage analysis at the object code level (or, possibly the assembly language code level) instead of at the traditional source code level. Others have proposed that object code level structural coverage analysis is applicable for Level B and C applications as well.

DO-248B/ED-94B [3], Frequently Asked Question (FAQ) #42 states that coverage analysis of the object code can be used *as long as analysis can be provided which demonstrates that the coverage analysis conducted at the object code will be equivalent to the same coverage analysis at the source code level. In fact, for Level A software coverage, DO-178B/ED-12B Section 6.4.4.2b states that if "...the compiler generates object code that is not directly traceable to Source Code statements. Then, additional verification should be performed on the object code..." This is often satisfied by analyzing the object code to ensure that it is directly traceable to the source code. Hence, DO-178B/ED-12B determines the conditions for analysis of the source code for*

NOTE: This position paper has been coordinated among the software specialists of certification authorities from the United States, Europe, and Canada. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.

structural coverage, and it does not prevent one from performing analysis directly on the object code [3].

FAQ #42 also states that the compiler may be utilized to simplify the object code analysis: *When utilizing compiler features to simplify analysis, one relies on the compiler to behave as expected. Therefore, one may need to qualify the compiler features being used as a verification tool (reference Section 12.2.2).* Some examples of such qualification of compiler features are provided below:

- Qualify that the compiler checks that all parameters passed are of the correct type, all array indexes are within bounds, etc. in order to preclude these items from the code review checklist.
- Qualify that the compiler does not add untraceable object code, in order to avoid performing the source code to object code traceability analysis activity.

Note: Neither of these qualification approaches are trivial; they merely provide examples of how aspects of the compiler may be qualified to satisfy the relevant verification objectives.

DO-248B/ED-94B, Discussion Paper (DP) #12, *Object Code to Source Code Traceability Issues*, also provides some insight into the subject. DP #12 focuses on source code to object code traceability; however, just as code optimization, compiler switch options, and inserted code by the compiler cause problems for source code to object code traceability, they also complicate the structural coverage at the object code level [2].

3.0 Motivation for Structural Coverage at the Object Code Level

Coverage at the object code level has become desirable because:

- Structural coverage analysis tools have advanced and can support this approach;
- Advances have been made in automating requirements-based testing and gathering structural coverage information as the tests are executed on the object code;
- Source code to object code traceability can be difficult and inconsistent;
- If done properly, it can demonstrate full code coverage at the object/assembly/machine code level;
- It can support more “valid” coverage as the testing and coverage analysis are conducted on an “abstraction” of the code that is closer to the final airborne software to be installed than the source code;
- It can be implemented with source code programming language independence;
- It can reduce time-consuming manual analysis; and

NOTE: This position paper has been coordinated among the software specialists of certification authorities from the United States, Europe, and Canada. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.

- It may not require instrumentation.

4.0 Certification Concerns

To date, manufacturers have proposed different approaches to demonstrate the adequacy of object code coverage (i.e., to show that it is at least as robust as the traditional source code coverage, supplemented with source code to object code traceability). Some of the approaches that have been proposed are to:

- Impose strict design and coding rules (e.g., prohibited features or constructs, language grammar rules, complexity restrictions, etc.) that are enforced by design and code review checklists. These rules are often hard to enforce and may cause maintenance issues later.
- Demonstrate and essentially qualify portions of the compiler. This analysis is often very dependent on the compiler settings and optimization.
- Perform additional object code review or analysis to ensure prohibited constructs are not used (i.e., design and coding rules have been followed), and/or that the compiler behaved as expected.
- Use a combination of the approaches above.

In most approaches to structural coverage analysis at the object code level, many assumptions are being made about the compiler operation (e.g., order of evaluations being left to right). It is often assumed that the compiler uses short-circuit logic (see section 3.6 of the FAA's MC/DC tutorial for more information [4]). However, the verification of those assumptions is typically not clear. Some developers propose tight coding standards; however, it is doubtful that every expression needed to code a system can be expressed in the manner assumed by these standards. For example, a case statement or sine or cosine function at the source code level might cause the compiler to generate a jump table or some similar structure. But then there is no criteria in most approaches to analyze those jump tables or other structures.

Basically, assumptions about the output of the compiler are being made. However, typically the developer performs no source code to object code traceability nor performs design reviews of the compiler functionality (nor qualifies those compiler functions) to determine that the compiler works as assumed. The behavioral abstractions of the compiler are unknown and are not addressed by most approaches. The logical abstractions at the source code level might not be present at the object code level which makes it more difficult to trace the object code to the source code.

5.0 Certification Authorities Software Team (CAST) Position

When structural coverage analysis at the object (assembly) code level is proposed for compliance to the DO-178B/ED-12B structural coverage objectives, the applicant should demonstrate that the coverage analysis at the object code level and source code level provide the same level of assurance. The following issues should be addressed (as a minimum):

- The approach should generate the same minimum number of test cases as that needed at the source code level appropriate to the software level of the application (e.g., MC/DC for Level A, decision coverage for Level B).
- The test cases used for coverage should be generated from the requirements (e.g., “structural testing” with module tests based on the code structure should not be used).
- All design and coding rules used to enforce the equivalence should be included in the design and coding standards, verification checklists, etc. and strictly followed.
- Data should be provided to substantiate all assumed compiler behavior (e.g., short-circuit forms with the appropriate optimization settings). (Note: In some cases, when compilers aggressively optimize or use self-optimization the behavior becomes unpredictable.)
- Analysis of the object code or qualification of a tool may be necessary to ensure that design and coding rules were followed and that the compiler performed as expected.
- Traceability between object code, source code, design, and requirements should exist.
- Architecture and complexity limitations should be documented and followed (e.g., number of nested *if*'s, number of conditions in a decision, nested function calls, etc.).
- The approaches for data coupling analysis and control coupling analysis should be performed by the applicant/developer, whether the coverage is performed on the linked object code or not.
- Data should be available to substantiate any object code not covered.
- The following questions should also be addressed (these are areas that are known to cause problems for many of the object coverage approaches):
 - How are parentheses addressed?
 - How are conditional calls addressed (e.g., jumps in branches)?

- How are long jump and long throw addressed (do they allow multiple entries and exits)?
- Are functions limited to only one entry point?
- Where does transfer of control in the function occur? (Note: It should typically be at the beginning.)
- How is control from outside to inside a function addressed?
- How are jump statements addressed (e.g., break, continue, return, goto)?
- Are bitwise operators used as Boolean operators prohibited?
- Do functions of the compiler (e.g., pre-parser) need to be qualified for the proposed compiler options and optimizations intended to be used?
- Is analysis of the object code needed to ensure design and coding rules were followed and that the compiler behaved as expected?
- Is structural coverage at the appropriate level also achieved for compiler-provided library functions and run-time system-provided library functions used or included in the airborne application?
- Does the linkage editor (and linking and loading procedures) have the capability to link into the application only those components and functions to be used and for which structural coverage has been achieved, or are unused (dead code) components and functions linked into the application also?
- Should linker or loader functions be qualified?

6.0 References

- [1] CAST-12 paper, entitled *Guidelines for Approving Source Code to Object Code Traceability*, completed June 2002.
- [2] RTCA/DO-178B and EUROCAE/ED-12B, *Software Considerations in Airborne Systems and Equipment Certification*.
- [3] RTCA/DO-248B and EUROCAE/ED-94B, *Final Report for Clarification of DO-178B/ED-12B “Software Consideration in Airborne Systems and Equipment Certification.”*
- [4] Hayhurst, Kelly et al, *A Practical Tutorial on Modified Condition/Decision Coverage*, May, 2001.