Object Code Verification and DO-178C Objective A7-9

By
Steve Morton

www.ldra.com

© LDRA Ltd. This document is property of LDRA Ltd.
Its contents cannot be reproduced, disclosed or utilised without company approval.

Introduction

Level A software developers have been tasked with the verification of object code that is directly untraceable to source code since the introduction of RTCA/DO-178B in 1992. This type of object code consists of executable statements that "[introduce] branches or side effects that are not immediately apparent at the Source Code level" [DO-178C 6.4.4.2.b note], including such things as compiler-generated array index boundary checks.

DO-178C corrected what many saw as an oversight by including this so-called "hidden objective" in Table A-7 of Annex A. The requirement to perform the activity remained the same, but was finally made explicit in the Annex tables.

But DO-178C also opened the door for using newer techniques for identifying and verifying object code not directly traceable to source code.

History of the "Hidden Objective"

As discussed extensively during RTCA Special Committee SC-205, DO-178C Annex A Table A-7 Objective #9 is not really "new". It is simply the revelation of one of DO-178B's "hidden objectives", which were called out in the body of the document but were omitted from specific mention in Annex A. As such, it should not be considered controversial, even though its addition was considered necessary due to instances of industry resistance to its substance.

DO-178C Objective A7-9 reads "Verification of additional code, that cannot be traced to Source Code, is achieved. (Ref. 6.4.4.c; Activity 6.4.4.2.b)"

Before delving into the practical effects of the revealed "hidden objective", it is instructive to first consider the textual basis on which it is constructed.

DO-178C Section 6.4.4 Updates

The wording of DO-178B and DO-178C section 6.4.4 are shown side-by-side below.

DO-178B§6.4.4	DO-178C§6.4.4
Test coverage analysis is a two step process, involving requirements-based coverage analysis and structural coverage analysis. The first step analyzes the test cases in relation to the software requirements to confirm that the selected test cases satisfy the specified criteria. The second step confirms that the requirements-based test procedures exercised the code structure. Structural coverage analysis may not satisfy the specified criteria. Additional guidelines are provided for resolution of such situations as dead code (subparagraph 6.4.4.3).	Test coverage analysis is a two step process involving requirements-based coverage analysis and structural coverage analysis. The first step analyzes the test cases in relation to the software requirements to confirm that the selected test cases satisfy the specified criteria. The second step confirms that the requirements-based test procedures exercised the code structure to the applicable coverage criteria. If the structural coverage analysis showed the applicable coverage was not met, additional activities are identified for resolution of such situations as dead code (see 6.4.4.3) The objectives for test coverage analysis are: a. Test coverage of high-level requirements is achieved. b. Test coverage of low-level requirements is achieved. c. Test coverage of software structure to the appropriate coverage criteria is achieved. d. Test coverage of software structure, both data coupling and control coupling, is achieved.

The existing text of DO-178B is not significantly changed in the updated verbiage of DO-178C. The two linguistic modifications are itemized below:

1. The “specified criteria” for structural coverage (based on the software level) is now described as the “applicable coverage criteria”. This is a simple wording change.
2. The “additional guidelines” of DO-178B §6.4.4.3 are now described as “additional activities” of DO-178C §6.4.4.3 as part of the specific effort to eliminate use of the term ‘guideline’.

The remaining text added enumerates the objectives of test coverage analysis as part of the overall restructuring of DO-178C to conform to an “objective-activity” format, where the objectives are formally specified and the activities to be used in satisfying the objectives identified in a systematic manner.

DO-178C Section 6.4.4.2.b Updates

DO-178C Objective A7-9 references the activities identified in §6.4.4.2 item b. The text of 6.4.4.2 item b from both DO-178B and DO-178C are shown below.

DO-178B §6.4.4.2.b	DO-178C §6.4.4.2.b
The structural coverage analysis may be performed on the Source Code, unless the software level is A and the compiler generates object code that is not directly traceable to Source Code statements. Then, additional verification should be performed on the object code to establish the correctness of such generated code sequences. A compiler-generated array-bound check in the object code is an example of object code that is not directly traceable to the Source Code.	Structural coverage analysis may be performed on the Source Code, object code, or Executable Object Code. Independent of the code form on which the structural coverage analysis is performed, if the software level is A and a compiler, linker, or other means generates additional code that is not directly traceable to Source Code statements, then additional verification should be performed to establish the correctness of such generated code sequences. Note: ”Additional code that is not directly traceable to Source Code statements” is code that introduces branches or side effects that are not immediately apparent at the Source Code level. This means that compiler-generated array-bound checks, for example, are not considered to be directly traceable to Source Code statements for the purposes of structural coverage analysis, and should be subjected to additional verification.

It should be noted that the purpose of the prescribed “additional verification” from both texts is to establish the “correctness of such generated code sequences”. This is to say that the introduced code is not required to be structurally covered necessarily, but rather the code is required to be verified to be correct with respect to the design in which it is introduced. For an array index boundary check, for example, it would be better if the introduced object code could not be executed because the design itself precludes violating the array’s bounds.

Instead, the guidance expects that two items will be fulfilled:

1. The fact that the object code is introduced is identified, and
2. The form and function of the code is verified (which may be done by review, analysis, or test).

Form and function, for the example of an array bound check, consists of ensuring that the limits put in the introduced object code correspond to the correct limits to the array, and checking that the correct index variable is compared to those limits.

Identifying the introduced object code is a bit more complex.

Identifying Untraceable Object Code

DO-148C Discussion Paper number 12 (§4.12) addresses "Object Code to Source Code Traceability Issues". In §4.12.2.2 "Approach to Traceability Analysis" a couple of methodologies are described:

> One approach to this analysis is to produce some code with fully representative language constructs of the application (e.g., case statements, if-then-else statements, loops) and to analyze the resultant object code. Also, it is important that the individual constructs are completely representative of the constructs used in the target object code build, including complex structures (e.g., nested routines). In some cases, object code may be data type dependent. The choice of the representative code constructs should be agreed with the appropriate certification authority. The link map should be examined to ensure that no unexpected library components are being incorporated into the target object code build.
>
> Another valid approach is to examine the target object code build.

The most commonly used approach is described in the first quoted paragraph above. A representative sampling of constructs is used to demonstrate that there are no untraceable object code statements introduced, or at least to identify what those introduced statements are. By doing so, areas that require additional attention during verification of the developed software are identified. The "representative" nature of the sampling has to be confirmed once the final software is completed, to ensure that the presence or absence of untraceable object code is known.

This representative source-to-object analysis is often performed by hand, adding to the difficulty and expense of this approach.

The representative approach also introduces two other complicating factors:

1. It ties the knowledge of untraceable object code to the specific compiler, including the settings used. This fact is often used as an argument against updating the compiler, even given potential issues that militate for a change.
2. It assumes that the compiler will always translate the source code constructs in the same manner, no matter the combinations of constructs used. While this is true of ideal compilers, assumptions are associated with risk.

The Certification Authorities Software Team (CAST) in their CAST-12 Position Paper entitled "Guidelines for Approving Source Code to Object Code Traceability" (completed in December 2002) identifies acceptable methods for performing the source-to-object analysis. In section 6 (Procedures), item h(1) states:

> (1) Manual review/analysis of the complete program - The source code and associated assembly code listings of the object code may be manually examined (reviewed) to detect any code added by the compiler beyond what is required for execution of the source code statements. The certification authority should examine the analysis/review results and some representative source code/object code groups to gain confidence that the analysis/review was done correctly and that the additional, untraceable object code identified correctly implements its required functionality.

It is important to note that in this paragraph use of assembly code listings are identified as an acceptable surrogate for object code, which is much more difficult to read.

The last quoted paragraph from DO-248C §4.12, however, opens the door to using object-code level verification in lieu of performing this extensive (and sometimes expensive) source-to-object code analysis.

Object Code Verification

As illustrated above, DO-178C §6.4.4.2.b contains a change from DO-178B that clarifies the viability of object code verification, beyond source-to-object analysis. Whereas DO-178B explicitly permitted source code level structural coverage analysis (while omitting mention of any other), DO-178C states that structural coverage analysis "may be performed on the Source Code, object code, or Executable Object Code". This explicit inclusion bolsters the argument that object code level verification can be used to help satisfy Objective A7-9.

Object code verification is characterized by the measurement of structural coverage at the object code level. That is, a tool is used to determine if the object code instructions are exercised by the applied requirements-based tests.

But object level coverage is not always a straightforward representation of the three statement-based structural coverages mandated by DO-178C. This is especially true for Modified Condition/Decision Coverage (MC/DC). Because MC/DC is predicated on pairings of condition sets showing independent effect on the overall expression's outcome for each condition, defining how to translate MC/DC into object code coverage can be difficult. Use of a tool that supports both source code and object code verification, therefore, offers a better approach to the issue.

Further, the ability to view the source code and object code from the application side-by-side fosters the capability to apply "additional verification" for object code not directly traceable to source code. Instead of performing a specific analysis of representative source code constructs, then scouring the application's object code for instances where untraceable object code is inserted, the tool directly supports viewing the instances that spawn such inserted object code, allowing a reviewer to verify such things as the correct application of array index boundaries.

Conclusions

DO-178C explicitly recognizes the validity of object code based measurement of structural coverage. Use of this technique can ease the burden of applying additional verification to object code not directly traceable to source code as required by Objective A7-9, especially when the structural coverage measurement tool employed supports side-by-side viewing of source code and object code coverage.

SGS TÜV SAAR FUNKTIONALE SICHERHEIT GEPRÜFT FUNCTIONAL SAFETY APPROVED

bsi. ISO 9001 Quality Management

LDRA Ltd. reserves the right to change any specifications contained within this literature without prior notice.

© 2016 LDRA Ltd

www.ldra.com

LDRA

LDRA UK & Worldwide
Portside, Monks Ferry,
Wirral, CH41 5LH
Tel: +44 (0)151 649 9300
e-mail: info@ldra.com

LDRA Technology Inc.
2540 King Arthur Blvd, Suite #228, Lewisville, Texas, 75056
Tel: +1 (855) 855 5372
e-mail: info@ldra.com

LDRA Technology Pvt. Ltd.
Unit B-3, Third floor Tower B, Golden Enclave,
HAL Airport Road, Bengaluru, 560017
Tel: +91 80 4080 8707
e-mail: india@ldra.com