EAC Decision on Request for Interpretation 2010-03
(Data Load)
2005 VVSG, Volume II, Section 5.4
2005 VVSG, Volume II, Section 5.4.2.a - 5.4.2.v

Date:
June 14, 2010

Question(s):
Shall database definition files be reviewed as source code under the guidelines found in section 5, Software Testing?

Section of Standards or Guidelines:
5.4 Source Code Review
The accredited test lab shall compare the source code to the vendor's software design documentation to ascertain how completely the software conforms to the vendor's specifications. Source code inspection shall also assess the extent to which the code adheres to the requirements in Volume I, Section 5.

5.4.2 Assessment of Coding Conventions
The accredited test lab shall test for compliance with the coding conventions specified by the vendor. If the vendor does not identify an appropriate set of coding conventions in accordance with the provisions of Volume I, Subsection 5.2.6, the accredited test lab shall review the code to ensure that it:

a. Uses uniform calling sequences. All parameters shall either be validated for type and range on entry into each unit or the unit comments shall explicitly identify the type and range for the reference of the programmer and tester. Validation may be performed implicitly by the compiler or explicitly by the programmer.

b. Has the return explicitly defined for callable units such as functions or procedures (do not drop through by default) for C-based languages and others to which this applies, and in the case of functions, has the return value explicitly assigned. Where the return is only expected to return a successful value, the C convention of returning zero shall be used or the use of another code justified in the comments. If an uncorrected error occurs so the unit must return without correctly completing its objective, a non-zero return value shall be given even if there is no expectation of testing the return. An exception may be made where the return value of the function has a data range including zero

c. Does not use macros that contain returns or pass control beyond the next statement

d. For those languages with unbound arrays, provides controls to prevent writing beyond the array, string, or buffer boundaries

e. For those languages with pointers or which provide for specifying absolute memory locations, provides controls that prevent the pointer or address from being used to overwrite executable instructions or to access inappropriate areas where vote counts or audit records are stored

f. For those languages supporting case statements, has a default choice explicitly defined to catch values not included in the case list

g. Provides controls to prevent any vote counter from overflowing. Assuming the counter size is large enough such that the value will never be reached is not adequate

h. Is indented consistently and clearly to indicate logical levels

i. Excluding code generated by commercial code generators, is written in small and easily identifiable modules, with no more than 50% of all modules exceeding 60 lines in length, no more than 5% of all modules exceeding 120 lines in length, and no modules exceeding 240 lines in length. “Lines” in this context, are defined as executable statements or flow control statements with suitable formatting and comments. The reviewer should consider the use of formatting, such as blocking into readable units, which supports the intent of this requirement where the module itself exceeds the limits. The vendor shall justify any module lengths exceeding this standard

j. Where code generators are used, the source file segments provided by the code generators should be marked as such with comments defining the logic invoked and, if possible, a copy of the source code provided to the accredited test lab with the generated source code replaced with an unexpanded macro call or its equivalent

k. Has no line of code exceeding 80 columns in width (including comments and tab expansions) without justification

l. Contains no more than one executable statement and no more than one flow control statement for each line of source code

m. In languages where embedded executable statements are permitted in conditional expressions, the single embedded statement may be considered a part of the conditional expression. Any additional executable statements should be split out to other lines

n. Avoids mixed-mode operations. If mixed mode usage is necessary, then all uses shall be identified and clearly explained by comments

o. Upon exit() at any point, presents a message to the user indicating the reason for the exit()

p. Uses separate and consistent formats to distinguish between normal status and error or exception messages. All messages shall be self-explanatory and shall not require the operator to perform any look-up to interpret them, except for error messages that require resolution by a trained technician

q. References variables by fewer than five levels of indirection (i.e., a.b.c.d or a[b].c->d)

r. Has functions with fewer than six levels of indented scope, counted as follows:

```
int function()
{
    if (a = true)
    {
        if ( b = true )
        {
            if ( c = true )
            {
                if ( d = true )
                {
                    while(e > 0 )
                    {
                        Code
                    }
                }
            }
        }
    }
}
```

s. Initializes every variable upon declaration where permitted

t. Has all constants other than 0 and 1 defined or enumerated, or shall have a comment which clearly explains what each constant means in the context of its use. Where “0” and “1” have multiple meanings in the code unit, even they should be identified. Example: “0” may be used as FALSE, initializing a counter to zero, or as a special flag in a non-binary category

u. Only contains the minimum implementation of the “a = b ? c : d” syntax. Expansions such as “j=a?(b?c:d):e;” are prohibited

v. Has all assert() statements coded such that they are absent from a production compilation. Such coding may be implemented by ifdef()s that remove them from

or include them in the compilation. If implemented, the initial program identification in setup should identify that assert() is enabled and active as a test version

Conclusion:

For clarification purposes, there is an error in VVSG Volume II Section 5.4.2. The requirements listed in VVSG Volume I Section 5.4.2 were intended to be labeled *a* through *v*, and should be referred to as such from this point forward.

In this RFI, the applicability of Data Definition Language (DDL) and Data Markup Language (DML) to VVSG Volume I Section 5 Software has been questioned. The following two points will be addressed:

- Must DDL and DML be subject to Volume II Section 5.4?
- Must DDL and DML be subject to Volume II Section 5.4.2.a thru 5.4.2.v?

Volume II Section 2.5.7.2.d states:

The vendor shall provide the following information:
If the software module or unit consists of, or contains, procedural commands (such as menu selections in a database management system for defining forms and reports, online queries for database access and manipulation, input to a graphical user interface builder for automated code generation, commands to the operating system, or shell scripts), a list of the procedural commands and reference to user manuals or other documents that explain them.

In order to support the evaluation required in VVSG Volume II Section 2.5.7.2.d, the manufacturer's documentation shall clearly specify:

1. If the DDL and DML presented for evaluation are using scripts, macros or other executable code.
2. If the DDL and DML could modify the results reported by modifying the database schema.

Volume II Section 5.4 requires the VSTL to ascertain how completely the software design documentation submitted by the manufacturer conforms to the manufacturer's specifications. Thus, DDL and DML shall be subject to Volume II Section 5.4. The VSTL shall thoroughly examine DDL and DML for logic including, but not limited to, triggers, functions, or stored procedures.

Secondly, DDL and DML shall be subject to Volume I Section 5.2.6:

Voting system software shall adhere to basic coding conventions. The coding conventions used shall meet one of the following conditions:

a. The vendors shall identify the published, reviewed, and industry-accepted coding conventions used and the accredited test lab shall test for compliance

b. The accredited test lab shall evaluate the code using the coding convention requirements specified in Volume II, Section 5

These guidelines reference conventions that protect the integrity and security of the code, which may be language-specific and language-independent conventions that significantly contribute to readability and maintainability. Specific style conventions that support economical testing are not binding unless adopted by the vendor.

Therefore, if a manufacturer has chosen to adhere to a coding convention outside of the VVSG 2005 then DDL and DML are exempt from Section 5.4.2.a thru 5.4.2.v.

Effective Date:

For all systems without an approved test plan.