U. S. Election Assistance Commission
Voting System Testing and Certification Program
1201 New York Avenue, NW, Suite 300
Washington, DC. 20005

September 23, 2010

Subject: EAC Response to Comments received for RFI 2010-07 Module length

---

1. Purpose

The purpose of this document is to respond to comments issued by registered manufacturers and accredited VSTLs with regard to DRAFT RFI 2010-07 Module Length. The responses listed in this document shall not be used as additional interpretations to the VSS, VVSG, RFI or NOC.

2. Standard being interpreted

2005 VVSG Volume II 5.4.2.i Assessment of Coding Conventions

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

3. Responses

3.1 SLI Global Solutions

Thank you for the opportunity to respond. We agree with the interpretation and have no other comments at this time.

3.1.1 EAC Response

Thank you for your comment

3.2 iBeta

iBeta has interpreted the sentence:

"Lines" in this context, are defined as executable statements or flow control statements with suitable formatting and comments to mean that a line = (executable statement or flow control statement) with suitable formatting and comments.

Based on the code we have reviewed, we believe that this has been the standard interpretation dating from the FEC development of the standards. Any other interpretation is going to:

a) penalize the programmer for utilizing white space to enhance the readability of the code

b) penalize the programmer for adding comments to clarify the module code

c) encourage programmers maintaining the code to sacrifice existing comments in modified modules that are close to the proposed interpretation of their line limit

d) force programmers maintaining code that is close to the proposed interpretation of the line limit to break up the module into smaller modules.

While item (d) is a reasonable expectation for new code, the cost of breaking up maintained code into sub-modules could decrease maintainability because the parameters required to do so might have to be global or passed by pointer in the C type languages. That in turn impacts the overhead in checking for pointer validity and array boundary sizes at the entry point to a module. In some cases that in turn might effect the speed of the module which may be a critical performance factor.

To meet this interpretation, the first thing the vendor is likely to do is "automatically" strip out white space from their modules which will make the current code less readable. Given a reasonable module with 4-6 parameters, a reasonable header describing that module could easily exceed 40 lines. (2 lines per parameter, 6 lines for the module purpose, 2 lines for outputs, 1-2 lines for error conditions, and 0-2 lines for side effects, 0-2 lines for files accessed, 1-20 lines for history, 12 lines for modules called. That leaves about 10 lines of execution if the industry best practice recommendation is 1 line of comment for each executable line of code. (At least 50% of the time, 60 lines per module). So as reviewers, all we will be looking at is headers. That is counter-productive, both to the reviewer and the coder.

Given the above, the next thing the vendor is likely to do is compress their history. This is a counter-productive activity lowering maintainability. Next they will compress the sequence of modules called into a single line which will in turn make that set of lines more difficult for the reviewer to read and check.

This interpretation penalizes vendors using newer languages and tools providing automatic generation of code (for example, Doxygen and Doc++) because these languages embedded in the comments tend to consume a number of lines which according to the proposed interpretation would further penalize the vendors and programmers that use it.

The penalty is not so extreme if the interpretation is amended so as not to include the header of the module. However this will impact the way iBeta currently performs the task for 60% of the code we review. We have tools that automatically count those lines of code. Now given the proposed interpretation, the reviewer will have to perform a subtraction for each module (including or excluding the header). We would need to research whether our current tools, some of which provide cyclomatic complexity numbers, would be useful at all under these circumstances.

While this interpretation appears to come directly from an apparent attempt to convert the "with" to an "or," in the stated requirement, it contains no industry references supporting the proposed interpretation of a "line" of code. If industry were to decide that a "line of code" includes comments, then any commenting of code would be counterproductive because one useful code metric is the number of defects or bugs per line of code and the addition of a comment would necessarily change that ratio depending on the ratio of comment statements, and "white space statements" to executable statements. The proposed metric will now depend on the style of the programmer. Thus even within any given organization the metric becomes meaningless. We believe with this interpretation the EAC is mandating that the VSTL's produce a code metric that cannot be compared within an organization or among different organizations. In other words the proposed measure renders this useless as a metric of any sort.

Will this interpretation make code more readable or maintainable? iBeta believes it will not. As such we are of the opinion that the following interpretation should be considered by the EAC: A line of code is defined as an executable statement or a flow control statement. A single executable statement or flow control statement with any supporting comments or formatting is counted as a single executable statement or flow control statement.

3.3 Unisyn Voting Solutions

Just to confirm here. The actual method header comments are NOT included in the line count. Method lines counts start with method declaration (e.g. public static void doSomethingElectionRelated (String inElectionID, String inBallotstyleID) throws IOException {) and ends with the final "}" All lines between outside these boundaries do not count towards the line count of the particular method, so that units called, method history, etc are exempt from being counted as "Lines".

Is this correct?

3.4 Dominion Voting Systems

Dominion Voting Systems notes with dismay the Conclusions of the draft RFI regarding how lines of code are to be counted. The conclusions asserted in the draft go against recognized coding practices and metrics and contravene the objective in VVSG to maximize code readability and clarity. The Conclusions also represent an unneeded and unwarranted shift from counting logical lines of code to physical lines of code.

Before discussing those Conclusions, Dominion wishes to state that the VVSG seems quite clear in this matter. Volume II, 5.2.4.i defines lines of code. Unless one wishes to redefine "with" as an "and", the definition points to logical lines of code, which is a standard method for determining lines of code for both review/measurement and software project management purposes. "With" means containing and adjacent to; and since formatting and comments are required by the coding languages themselves and VVSG, this meaning of "with" is sensible and passes the plainface test when reading statutory and similar language. The writers of VVSG also noted that comments and formatting might cause modules to exceed the prescribed limits: "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." Note the "module itself" wording. It shows that the VVSG authors believed that formatting should not be considered part of the line count.

Recognized coding standards and metrics: From the NASA document "Quantitative Software Management" (http://software.gsfc.nasa.gov/docs/QSM-class/Day%201-Cost/04a-Size.ppt)

- Typically either physical lines or logical executable lines are used when counting SLOC
- Comments and blanks should never be included in any lines of code count

Bringing comments, formatting, and blank lines into the lines of code count is not required by VVSG. Similarly, what problem is counting logical lines of code (as has been done by at least some of the VSTLs for years) causing to the readability and clarity of voting systems code or to the performance of certified voting systems? Is the change to established practice mandated by the draft Conclusions a solution looking for a problem?

Readability of Code:

Bringing comment lines into the module size limits contravenes the obvious push by the authors of VVSG to code readability. Numerous clauses in both VVSG volumes, as well as sub-clauses a through w for Volume II 5.2.4 speak to this heavily implied requirement. There is no reason to publish an Interpretation that will cause code developers to limit formatting and comments to stay under an artificially lowered module size limit.

Furthermore, header blocks, which include change histories and other information, can become quite lengthy in a well managed system: a system where every change is noted. In addition, many modern tools exist to leverage xml tagging within comments to autogenerate specification, change, and data-dictionary documents. These xml tags will further increase LOC totals if comments are counted- forcing developers to abandon these tools thus sacrificing their use for reducing counts.

In conclusion, the draft Conclusions turn the "with" of Volume II, clause 5.2.4.i into an "and". This RFI, if published as written, will penalize developers for writing the best documented code. As a logical result, code readability in voting systems will diminish. This serves neither the letter and intent of VVSG nor the EAC's stakeholders – those entities

reviewing voting systems code. If all of the comment and formatting lines are to be counted there will often only be on the order of 10-15 lines of executable code. This actually negates the stated intent of single function, well-encapsulated modules (functions, methods, classes etc). Coherent, logical units will have to be broken up across several sub modules (4-6) instead of a single module of 60 executable lines of code. The Conclusions must be modified before publication of this RFI to exclude formatting and comments. If this RFI is published as drafted, the effectivity date will cause the systems in Certification to undergo some amount of code refactoring, causing delays and the refactoring of previously reviewed and tested stable code.

3.5 EAC Response to 3.2, 3.3, and 3.4

The VVSG and VSS Vol II 5.4.2.i states, “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.” The reviewer, meaning the VSTL, can judge whether the use of blank lines used for formatting will count toward the overall module line count where the module exceed the limits stated in the requirement. The VSTL’s decision to exempt blank lines from the line count shall be limited to instances where blank lines improve the readability of the module. Headers can be considered to fall under this same exception. Additionally, the manufacture may submit, in writing, to the EAC a justification for exceeding the limits stated in the requirement. The EAC will rule upon these requests on an individual case by case basis.