ISO/IEC JTC 1/SC 22/OWGV N ~~0099~~0102, corrected

Annotations to N0099 made during Meeting #6 of OWGV

Date 3 October 2007, corrected 25 October 2007
Contributed by Derived from N0099
Original file name n0099.doc
Notes During the discussion of N0099 during Meeting #6, annotations were made to the document. These annotations are valuable enough to be captured. Accordingly, document N0102 was created. The annotations are shown as changes using "Track Changes".

Formatted: Indent: Left: 0 pt, Hanging: 108 pt
Formatted: Font: Not Bold
Formatted: Font: Not Bold

~~*Proposal to the ISO/IEC Project 22.24772: Guidance for Avoiding Vulnerabilities through Language Selection and Use*~~

~~**Date** 29 September 2007~~
~~**Contributed by** Larry Wagoner~~
~~**Original file name** sc22_owgv_safety_proposal_v3.doc~~
~~**Notes**~~

Part 1: Reason for the Proposal

Document N0073 was a derivation of the security vulnerabilities observed in the wild to actionable modifications in computer languages. The safety world, which overlaps considerably with the security world with respect to vulnerabilities, deserves a similar analysis.

This document will attempt a similar derivation from the real world to actionable modifications in computer languages. It is expected that there will be some or even considerable overlap with the observations seen in the security world. However, it is expected that there will be additional recommendations unique to the safety world or which have not been observed to a large enough degree in the security world.

Part 2: Derivation from Frequently Occurring Vulnerabilities to CWE

[Subsequent to Meeting #6 it was noted that this heading should read:
"Part 2: Derivation from a Measurement Based Safer Subset of ISO C"**]**

Formatted: Font: (Default) Nimbus Roman No9 L, (Asian) Nimbus Sans L, 14 pt, Bold

There doesn't seem to be a central safety collection of observed problems for software safety that is comparable to the CERT collection. There are a few popularly cited ones such as the Therac-25, NASA Mars missions and Ariane-5. There are also some blogs such as ACM Risks (http://catless.ncl.ac.uk/Risks) and IEEE Spectrum's The Risk Factor (http://blogs.spectrum.ieee.org/riskfactor/).

There are many safety guidelines in existence. Many, however, were generated with a "gut feel." Some are too vague or too "feel good" to be of practical use. Others are

heavily concentrated on requirements, design and documentation, all of which are outside of the scope of this project. Any guidance that does not have an underlying empirical basis is a distraction to the development of software and in the worst case can make software less reliable than it would have been without the guidance.

One that focuses on the computer language is the "Guidelines for the Use of the C Language in Vehicle Based Software" published by The Motor Industry Software Reliability Association (MISRA). MISRA-C was originally published in 1998 as a set of 127 guidelines for using C in safety critical systems. The guidance was updated in 2004, with the guidance renumbered, reorganized, modified and made consistent as a set of 141 rules. Vendors now sell tools that help organizations verify MISRA-C compliance. MISRA-C is based on ISO 9899:1990 as amended by ISO/IEC 9899/COR1:1995, AMD1:1995, and COR2:1996 which has been superseded by ISO/IEC 9899:1999. MISRA-C is a good basis for actionable language guidance for the safety community. For the purposes of SC22 OWGV, we will have to update and filter the rules to reflect the changes made in C99.

MISRA C has also undergone analysis by some researchers. One of note is Les Hatton, Chair in Forensic Software Engineering at the University of Kingston, UK. Hatton proposes in [1] dividing rules into two categories:

A – Rules that promote a common style

B – Rules that promote avoidance of programming features that are thought to or have caused problems

B rules can then be divided into two categories:

B.1 – Rules whose violation is capable of being the root cause of a problem, but of which an actual instance is lacking

B.2 – Rules whose violation can be shown to be the root cause at least one known failure

In [2] Hatton develops a small number of rules which avoid known faults. He bases them on the same ISO C standard as MISRA C (that is, ISO/IEC 9899: 1990 (C90) that includes the normative addedendum and two technical corrigenda from 1993-1995) and for simplicity he refers to this as C94.

He states five goals in developing these rules:

- Every rule is associated with faults which appeared in the quoted surveys in his paper
- Every rule covers as many of the fault modes as possible to reduce the total number of rules
- Rules are easy to understand and as unambiguous as possible
- Rules are as non-contentious as possible to ease acceptance
- The totality of the rules cover the vast majority of the faults described in earlier surveys

For the safety world, his rule development follows a similar pattern to how the rules were derived from the security world and proposed in SC22 OWGV document N0073. His

rules will be stated below and then a cross-mapping will be made to the templates created from N0073. Should any of his rules not be covered by an already existing template, then a new template will need to be created for consideration.

The list of twenty rules:

1. S_GLOB: There shall be no dependence on any of the undefined features of ISO C
   This refers to the 97 items in Appendix G of C94, such as "all automatics shall should be initialized before use," "do not divide by zero," and "expressions shall not depend on evaluation order."
2. F_PROT: All function calls and definitions shall be preceded by a new-style prototype
3. F_COMP: All function call arguments and function definition parameters shall be compatible with the corresponding arguments in the corresponding function declaration
   This would prevent a long being passed to a short or even to an unsigned short.
4. E_PREC: A boolean-valued sub-expression shall not appear next to a bit operator, an assignment operator or a relational operator in a if or while expression
   This is a spin on the "thou shall put parenthesis around all expressions which results in senseless situations (and programmer irritation with standards/guideline) such as:

```
x =  (x + 5); instead of x = x + 5;
```

   It narrows the focus to situations such as:

```
if (flags & bitmask != 0)
```

   where the programmer really meant

```
if ((flags & bitmask) != 0)
```

5. E_SIDE: Every expression statement shall have at least one side-effect for any execution path
   If a statement has no side-effects (i.e. modifying a file or accessing a volatile), the statement doesn't do anything. The classic example is:

```
i == j; where the programmer meant i= j;
```

6. E_COMM: The expression on the left hand side of a comma operator shall have at least one side-effect
   This is to avoid code such as:

```
z = i++ + (i,y++);
```

   which is the same as

```
z = i++ + y++;
```

Therefore it begs the question as to why the programmer used (i,y++) and thus should be flagged.
7. E_UNEG: An unsigned expression shall not be compared for negativity
8. E_CSIGN: No implicit conversion shall change the signed nature of an object or reduce the number of bits in that object
9. E_REACH: Every non-null statement shall be reachable
   This is stated this way to avoid tagging situations such as:

```
case YES:
    …
    return;
    break;          /* unreachable, but who cares? */
```

10. E_HIDE: Local objects shall not hide objects with internal or external linkage

For example:

```
int i = 4;          /* i is 4 */
…
{
        i = 5;      /* i = 5 */
        …
}
                    /* i = 4 */
```

11. E_EXTR: The extern keyword shall not be used in a nested block
12. E_FEQL: Floating point objects shall not be compared for equality or inequality
13. E_FLOOP: Floating point objects shall not be used in the initialization, controlling or re-initialization expressions of a for statement or the controlling expression of a while statement.
14. E_BITF1: Every named bitfield of size 1 shall use the unsigned or signed keyword in its declaration
15. E_NARRW: No pointer shall be cast to a narrower integral type

For example:

```
int i;
char *c;
i = (int)c;
```

16. E_PCAST: Pointers shall not be cast to different pointer types
17. E_CASE1: A switch statement shall have at least one 'case:' and shall have exactly one 'default:'
18. E_CFALL: No 'case:' or 'default:' shall be reachable from the previous 'case:' or 'default:' except by direct fall-through

For example:

```
switch (i)
{
        case 1:
        case 2:         /* this is allowed - case 1 falling through to case 2 */
        ...
        case 6:
                ++k;
        case 7:         /* this shouldn't be allowed - case 6 falling through to */
                        /* case 7 -- it could be intended, but it is likely that the */
                        /* programmer left out the break statement in case 6 */
...
}
```

19. P_ARGS2: No function-like macro shall use an argument more than once

For example

```
#define SQR(x)  ((x)*(x))
…
z = SQR (y++);
```

This causes y to be incremented by 2 and the value of z is dependent on the evaluation order.

20. P_PAREN: All macro arguments shall be parenthesized unless by doing so a syntax violation would be created

The goal of SC22 OWGV is to be as language independent at possible. Each language has trade-offs to allow certain capabilities and freedoms (imagine writing an OS using COBOL, but COBOL is great for business applications). These freedoms at times come at the expense of safety or security. Hatton's work is oriented squarely at C. Some of his rules are very C specific. Other rules can apply to other languages, though modifications may be needed. For instance, Rule 6, E_COMM: The expression on the left hand side of a comma operator shall have at least one side-effect is very C specific. Rule 1, _GLOB: There shall be no dependence on any of the undefined features of ISO C, can easily be made language independent. Even though it could be made language independent, it does not mean that it would apply to all languages, but rather the subset of languages that have undefined features.

Table 1 contains a list of the rules along with a decision whether the rule is or can be made language independent. For those rules which are or can be language independent, the last column will contain either the words "As is" meaning that the rule as it is stated is language independent, or if it is language independent, the last column will contain a modified, language independent version of the rule.

Number	Abbreviation	Existing Rule	Potential for Language Independence	Use as is or modified rule
1	S_GLOB	There shall be no dependence on any of the undefined features of ISO C	Yes	There shall be no dependence on any of the undefined features of the language standard.
2	F_PROT	All function calls and definitions shall be preceded by a new-style prototype	Yes, in a broadened sense	Deprecated features of the language should not be used.
3	F_COMP	All function call arguments and function definition parameters shall be compatible with the corresponding arguments in the corresponding function declaration	Yes	As is
4	E_PREC	A boolean-valued sub-expression shall not appear next to a bit operator, an assignment operator or a relational operator in an if or while expression	Yes	As is

Number	Abbreviation	Existing Rule	Potential for Language Independence	Use as is or modified rule
5	E_SIDE	Every expression statement shall have at least one side-effect for any execution path	Yes	As is
6	E_COMM	The expression on the left hand side of a comma operator shall have at least one side-effect	No, too language specific	
7	E_UNEG	An unsigned expression shall not be compared for negativity	Yes, include in an integer coercion rule	Integer coercion, such as comparing an unsigned expression for negativity, shall not be done.
8	E_CSIGN	No implicit conversion shall change the signed nature of a object or reduce the number of bits in that object	Yes	As is
9	E_REACH	Every non-null statement shall be reachable	Yes	All statements should be reachable
10	E_HIDE	Local objects shall not hide objects with internal or external linkage	No, too language specific	
11	E_EXTR	The extern keyword shall not be used in a nested block	No, too language specific	
12	E_FEQL	Floating point objects shall not be compared for equality or inequality	Yes	As is
13	E_FLOOP	Floating point objects shall not be used in the initialization, controlling or re-initialization expressions of a for statement or the controlling expression of a while statement.	Yes	As is
14	E_BITF1	Every named bitfield of size 1 shall use the unsigned or signed keyword in its declaration	Yes	As is
15	E_NARRW	No pointer shall be cast to a narrower integral type	Yes	As is
16	E_PCAST	Pointers shall not be cast to different pointer types	Yes	As is

Number	Abbreviation	Existing Rule	Potential for Language Independence	Use as is or modified rule
17	E_CASE1	A switch statement shall have at least one 'case:' and shall have exactly one 'default:'	Yes	As is
18	E_CFALL	No 'case:' or 'default:' shall be reachable from the previous 'case:' or 'default:' except by direct fall-through	Yes	As is
19	P_ARGS2	No function-like macro shall use an argument more than once	No	
20	P_PAREN	All macro arguments shall be parenthesized unless by doing so a syntax violation would be created	No	

Table 1 : Mapping of Hatton [2] Rules to SC22 OWGV Templates

For those rules which cannot be made language independent, the rule is eliminated from consideration. For those rules which are or can be language independent, the last column will contain either the rule as it is, if it is language independent, or contain a modified, language independent version of the rule.

Table 2 contains the derived rules from Hatton's work and a cross mapping of the rules to the currently existing SC22 OWGV templates. New templates will need to be created for those for annotated as "New" and for which no template is listed. For those already covered by existing templates, the existing templates will need to be checked to be sure that the template fully reflects the rule and annotated to reflect the safety basis in the subsections such as the mechanism of failures and avoidance of the vulnerability.

Number	New Derived Rule	Template/New
1	There shall be no dependence on any of the undefined features of the language standard.	Broad theme, but partially covered by EWF, BQF, maybe FAB. Specifying particular instances as in EWF, BQF, and FAB would be more productive than just saying there should be no dependence on undefined features. So additional templates created from working from the bottom up will be partially cover this rule.
2	Deprecated features of the language should not be used.	New

Number	New Derived Rule	Template/New
3	All function call arguments and function definition parameters shall be compatible with the corresponding arguments in the corresponding function declaration	New
4	A boolean-valued sub-expression shall not appear next to a bit operator, an assignment operator or a relational operator in an if or while expression	New
5	Every expression statement shall have at least one side-effect for any execution path	add in XYQ
6	Integer coercion, such as comparing an unsigned expression for negativity, shall not be done.	add in XYE
7	No implicit conversion shall change the signed nature of a object or reduce the number of bits in that object	add in XYF
8	All statements should be reachable	add in XYQ
9	Local objects shall not hide objects with internal or external linkage	New
10	Floating point objects shall not be compared for equality or inequality	New
11	Floating point objects shall not be used in the initialization, controlling or re-initialization expressions of a for statement or the controlling expression of a while statement.	New
12	Every named bitfield of size 1 shall use the unsigned or signed keyword in its declaration	add into XYF
13	No pointer shall be cast to a narrower integral type	New
14	Pointers shall not be cast to different pointer types	New
15	A switch statement shall have at least one 'case:' and shall have exactly one 'default:'	New
16	No 'case:' or 'default:' shall be reachable from the previous 'case:' or 'default:' except by direct fall-through	New

Table 2 Mapping of Hatton [2] Rules to SC22 OWGV Templates

Of the sixteen rules in Table 2, six are covered by or could be incorporated directly into existing templates. Templates need to be created for the remaining ten items. It is

proposed that the remaining items be grouped into the following seven categories, each of which would correspond to a new template.

Deprecated features
2 Deprecated features of the language should not be used

Functions
3 All function call arguments and function definition parameters shall be compatible with the corresponding arguments in the corresponding function declaration

Boolean tests
4 A boolean-valued sub-expression shall not appear next to a bit operator, an assignment operator or a relational operator in an if or while expression

Scope issues
9 Local objects shall not hide objects with internal or external linkage

Floating point
10 Floating point objects shall not be compared for equality or inequality
11 Floating point objects shall not be used in the initialization, controlling or re-initialization expressions of a for statement or the controlling expression of a while statement

Casting of pointers
13 No pointer shall be cast to a narrower integral type
14 Pointers shall not be cast to different pointer types

Switch/case issues
15 A switch statement shall have at least one 'case:' and shall have exactly one 'default:'
16 No 'case:' or 'default:' shall be reachable from the previous 'case:' or 'default:' except by direct fall-through

The creation of the templates should be fairly straightforward for these seven categories.

Part 3: MISRA-C

Many of Hatton's rules are very similar or virtually identical to those in MISRA-C. However, there are many more rules in MISRA-C than the twenty that Hatton proposes. Therefore, it would be prudent to check the entire rule set of MISRA-C to determine whether there are additional rules or guidance of value that can be derived. The MISRA-C rules must be purchased and so cannot appear in this paper and can be only referred to by number. Fortunately a copy of the rules for personal use is relatively inexpensive as it costs only 10 pounds sterling (about 20 U.S. dollars) for an electronic PDF version emailed to you. The rules do take a couple of days after your order to arrive in your in-box.

The templates in SC22 OWGV are expected to be more encompassing than individual rules as presented in MISRA-C. Therefore, it would be expected that multiple MISRA-C rules would be covered by a single template. Hatton's rules are likewise broader, in general, than MISRA-C rules. The specifics covered by MISRA-C rules and other specifics should be described in the text of the templates if that is permissible under MISRA-C licensing guidelines.

Initially, a mapping of the MISRA-C rules to the derived rules in Table 2 will be made in Table 3. The remaining rules in MISRA-C will then be either grouped together into categories or rejected as out of scope, too language specific, etc. This mapping will demonstrate which rules are covered by the derived rules from Hatton's work and what other rules could be derived from the MISRA-C work. For MISRA-C rules that are within the scope of the rules from Table 2, but which are definitely not covered by the current version of Hatton's rule will be tagged with the words "add-in." Add-in will refer to all rules listed after the add-in tag.

Number	Derived Rule from Part 2	MISRA-C:2004 Rules	Template/New
2.1	There shall be no dependence on any of the undefined features of the language standard.	1.2 add-in 3.1, 3.2, 3.3, 4.1	Broad theme, but partially covered by EWF, BQF, maybe FAB.
2.2	Deprecated features of the language should not be used.	1.1	New
2.3	All function call arguments and function definition parameters shall be compatible with the corresponding arguments in the corresponding function declaration	8.3, 16.4 add-in 8.1, 8.4, 8.6, 8.7, 16.1, 16.3, 16.5, 16.6, 16.7, 16.9	New
2.4	A boolean-valued sub-expression shall not appear next to a bit operator, an assignment operator or a relational operator in an if or while expression	12.5, 12.6 add-in 13.2	New
2.5	Every expression statement shall have at least one side-effect for any execution path	13.1, 14.2 add-in 12.3, 12.4	add in XYQ
2.6	Integer coercion, such as comparing an unsigned expression for negativity, shall not be done.	12.9	add in XYE
2.7	No implicit conversion shall change the signed nature of a object or reduce the number of bits in that object	10.1, 10.2, 10.3, 10.4, 10.5 add-in 12.11	add in XYF
2.8	All statements should be reachable	14.1 add-in 2.4	add in XYQ

Number	Derived Rule from Part 2	MISRA-C:2004 Rules	Template/New
2.9	Local objects shall not hide objects with internal or external linkage	add-in 5.2, 8.9, 8.10	New
2.10	Floating point objects shall not be compared for equality or inequality Floating point objects shall not be used in the initialization, controlling or re-initialization expressions of a for statement or the controlling expression of a while statement.	13.3, 13.4 add-in 1.5, 12.12	New
2.11	Every named bitfield of size 1 shall use the unsigned or signed keyword in its declaration	6.4 add-in 3.5, 12.7	add into XYF
2.13	No pointer shall be cast to a narrower integral type Pointers shall not be cast to different pointer types	11.1, 11.2, 11.3, 11.4 add-in 11.5	New
2.14	A switch statement shall have at least one 'case:' and shall have exactly one 'default:' No 'case:' or 'default:' shall be reachable from the previous 'case:' or 'default:' except by direct fall-through	15.2, 15.3 add-in 14.8, 15.1, 15.4, 15.5	New

Table 3: Mapping of MISRA-C Rules/Advisories to Rules Derived in Part 2

Remaining rules that need to be covered are categorized in Table 4 along with the remaining rules with are out of scope, etc. The new rules that need to be covered, the existing template or the need for a new template will be indicated in the last column.

Category	MISRA-C Rule/Advisory	Template/New
Undefined behavior of shift operator	12.8	add-in EWF or XYY
Undefined behavior of increment (++) and decrement (--) operator	12.13	add-in EWF
Use of obscure language features	12.10	New
Control Flow – if structure	14.9, 14.10	New
Initialization of variables, arrays, structures and enumerated lists	9.1, 9.2, 9.3	New
Loop control	13.5, 13.6, 14.6	New
Operator Precedence	12.1, 12.2	add-in FAB

Functions – control flow/return values	16.2, 16.8, 16.10	New
Macros	19.4, 19.7,19.8, 19.9, 19.10, 19.11	New
Reuse of identifiers or reserved identifiers	5.3, 5.4, 5.5, 5.6, 5.7, 20.1, 20.2	add-in YOW
Overlapping or reuse of memory	18.2, 18.3	New
Restrictions on types	6.1, 6.2, 6.3, 6.5,	New
Preprocessor	14.3, 19.1	New
Compiler issues	1.3, 1.4, 3.4, 5.1	New
Pointers	17.1, 17.2, 17.3, 17.4, 17.5, 17.6	(maybe) add-in XYK or New
Libraries	20.3, 20.4	New
Tool use	21.1 (good rule, but where to put it?)	New
Inappropriate Rules (out of scope, too language specific, style guidance, etc.)	2.1, 2.2, 2.3, 3.6, 4.2, 7.1, 8.5, 8.8, 8.11, 8.12, 10.6~~, 14.4, 14.5, 14.7,~~ 18.1, ~~18.4, 19.2, 19.3, 19.5, 19.6, 19.12, 19.13, 19.14, 19.15, 19.16, 19.17,~~ 20.5, 20.6, 20.8, 20.9~~, 20.10~~, ~~20.11~~, 20.12	

Table 4: Categorization of Remaining MISRA-C Rules/Advisories

Incorporating Table 4 into Table 3 yields Table 5.

Number	New Derived Rule	MISRA-C:2004 Rules	Template/New
2.1	There shall be no dependence on any of the undefined features of the language standard.	1.2 add-in 3.1, 3.2, 3.3, 4.1	Broad theme, but partially covered by EWF, BQF, maybe FAB.
2.2	Deprecated features of the language should not be used.	1.1	New
2.3	All function call arguments and function definition parameters shall be compatible with the corresponding arguments in the corresponding function declaration	8.3, 16.4 add-in 8.1, 8.4, 8.6, 8.7, 16.1, 16.3, 16.5, 16.6, 16.7, 16.9	New
2.4	A boolean-valued sub-expression shall not appear next to a bit operator, an assignment operator or a relational operator in an if or while expression	12.5, 12.6 add-in 13.2	New

Number	New Derived Rule	MISRA-C:2004 Rules	Template/New
2.5	Every expression statement shall have at least one side-effect for any execution path	13.1, 14.2 add-in 12.3, 12.4	add in XYQ
2.6	Integer coercion, such as comparing an unsigned expression for negativity, shall not be done.	12.9	add in XYE
2.7	No implicit conversion shall change the signed nature of a object or reduce the number of bits in that object	10.1, 10.2, 10.3, 10.4, 10.5 add-in 12.11	add in XYF
2.8	All statements should be reachable	14.1 add-in 2.4	add in XYQ
2.9	Local objects shall not hide objects with internal or external linkage	add-in 5.2, 8.9, 8.10	New
2.10	Floating point objects shall not be compared for equality or inequality Floating point objects shall not be used in the initialization, controlling or re-initialization expressions of a for statement or the controlling expression of a while statement.	13.3, 13.4 add-in 1.5, 12.12	New
2.11	Every named bitfield of size 1 shall use the unsigned or signed keyword in its declaration	6.4 add-in 3.5, 12.7	add into XYF
2.13	No pointer shall be cast to a narrower integral type Pointers shall not be cast to different pointer types	11.1, 11.2, 11.3, 11.4 add-in 11.5	New
2.14	A switch statement shall have at least one 'case:' and shall have exactly one 'default:' No 'case:' or 'default:' shall be reachable from the previous 'case:' or 'default:' except by direct fall-through	15.2, 15.3 add-in 14.8, 15.1, 15.4, 15.5	New
2.15	Undefined behavior of shift operator	12.8	add-in EWF or XYY
2.16	Undefined behavior of increment (++) and decrement (--) operator	12.13	add-in EWF
2.17	Use of obscure language features	12.10	New
2.18	Control Flow – if structure	14.9, 14.10	New
2.19	Initialization of variables, arrays, structures and enumerated lists	9.1, 9.2, 9.3	New
2.20	Loop control	13.5, 13.6, 14.6	New

Number	New Derived Rule	MISRA-C:2004 Rules	Template/New
2.21	Operator Precedence	12.1, 12.2	add-in FAB
2.22	Functions – control flow/return values	16.2, 16.8, 16.10	New
2.23	Macros	19.4, 19.7,19.8, 19.9, 19.10, 19.11	New
2.24	Reuse of identifiers or reserved identifiers	5.3, 5.4, 5.5, 5.6, 5.7, 20.1, 20.2	add-in YOW
2.25	Overlapping or reuse of memory	18.2, 18.3	New
2.26	Restrictions on types	6.1, 6.2, 6.3, 6.5	New
2.27	Preprocessor	14.3, 19.1	New
2.28	Compiler issues	1.3, 1.4, 3.4, 5.1	New
2.29	Pointers	17.1, 17.2, 17.3, 17.4, 17.5, 17.6	(maybe) add-in XYK or New
2.30	Libraries	20.3, 20.4	New
2.31	Tool use	21.1 (good rule, but where to put it?)	New

Table 5: Results of Mapping MISRA-C:2004 Rules into Rules Derived in Part 2

Part 4: Joint Strike Fighter Air Vehicle Coding Standards

Another document that is designed to aid developers in creating safety critical software is the Joint Strike Fighter (JSF) Air Vehicle (AV) C++ Coding Standards for the System Development and Demonstration Program. This coding standard also bases many of their rules on MISRA-C. Although in this case, the rule set is based on the April, 1998 version of MISRA-C instead of the newer 2004 version. The MISRA-C:1998 rule set had 127 rules, of which 93 were required and 34 were deemed advisories. The JSF AV set of rules consists of 221 rules. Seventy four of the MISRA-C:1998 rules were used either in their original form, extended or revised to create the JSF AV rules. Two of the MISRA-C:1998 rules were combined to create one of the JSF AV rules, so of the 221 JSF AV rules, 73 are very close or essentially identical to MISRA-C:1998 rules/advisories.

Since the MISRA-C:2004 are a revised set of MISRA-C:1998 and those have been examined in Part 2, then only the rules that are not tagged in the JSF AV document with an associated MISRA-C rule will be examined. A similar categorization to that done in Table 4 will be presented in Table 6.

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6
2.1	There shall be no dependence on any of the undefined features of the language standard. (1.2, add-in 3.1, 3.2, 3.3, 3.4, 4.1)	210, 211, 212, 214	Broad theme, but partially covered by EWF, BQF, maybe FAB.	It should be localized to EWF, BQF, FAB. Discuss pragmas and assertions also.
2.2	Deprecated features of the language should not be used. (1.1, 4.2, 20.10)	8, 152	New	Create a new description for deprecated features, MEM. This might be focal point of a discussion of what to do when your language standard changes out from underneath you. Include legacy features for which better replacements exist. Also, features of languages (like multiple declarations on one line) that commonly lead to errors or difficulties in reviewing. The generalization is that experts have determined that use of the feature leads to mistakes.
2.3	All function call arguments and function definition parameters shall be compatible with the corresponding arguments in the corresponding function declaration (8.3, 16.4, add-in 8.1, 8.4, 8.6, 8.7, 16.1, 16.3, 16.5, 16.6, 16.7, 16.9)		New	Also discuss external linkage, cross-language calls, and API calls. Add into XZM.
2.4	A boolean-valued sub-expression shall not appear next to a bit operator, an assignment operator or a relational operator in an if or while expression (12.5, 12.6, add-in 13.2)		New	Add to the new JCW.

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6
2.5	Every expression statement shall have at least one side-effect for any execution path (13.1, 14.2, add-in 12.3, 12.4). Perhaps this should be phrased as statements that execute with no effect on all possible execution paths.		add in XYQ	We look at XYQ. XYQ concerns code that cannot be reached. That is somewhat different than code that executes with no result. The latter is a symptom of poor quality code but may not be a vulnerability. We should introduce a new item, KOA, for code that executes with no result because it is a symptom of misunderstanding during development or maintenance. (Note that this is similar to unused variables.) We probably want to exclude cases that are obvious, such as a null statement, because they are obviously intended. It might be appropriate to require justification of why this has been done. These may turn out to be very specific to each language. The rule needs to be generalized.
2.6	Integer coercion, such as comparing an unsigned expression for negativity, shall not be done. (12.9)		add in XYE	Added material to XYE
2.7	No implicit conversion shall change the signed nature of a object or reduce the number of bits in that object (10.1, 10.2, 10.3, 10.4, 10.5, add-in 12.11)	162	add in XYF	Added material to XYF
2.8	All statements should be reachable (14.1, add-in 2.4)	127	add in XYQ	Already covered by XYQ
2.9	Local objects shall not hide objects with internal or external linkage (add-in 5.2, 8.9, 8.10)	159	New	Add to YOW. Also add something about issues in redefining and overloading operators.

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6
2.10	Floating point objects shall not be compared for equality or inequality Floating point objects shall not be used in the initialization, controlling or re-initialization expressions of a for statement or the controlling expression of a while statement. (13.3, 13.4, add-in 1.5, 12.12)	184	New	Add to a new description PLF that says that when you use floating point, get help. The existing rules should be cross-referenced.
2.11	Every named bitfield of size 1 shall use the unsigned or signed keyword in its declaration (6.4, 6.5, add-in 3.5, 12.7)		~~add into XYF~~New	Write a new vulnerability description, STR, that deals with bit representations. It would say that representations of values are often not what the programmer believes they are. There are issues of packing, sign propagation, endianness and others. Boolean values are a particular problem because of packing issues. Programmers who depend on the bit representations of values should either utilize language facilities to control the representation or document that the code is not portable.
2.13	No pointer shall be cast to a ~~narrower~~ integral type Pointers shall not be cast to different pointer types (11.1, 11.2, 11.3, 11.4, add-in 11.5)	~~175~~, 182	New	We will write a new description, HFC, to cover pointer casting and pointer type changes.

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6
2.14	A switch statement shall have at least one 'case:' and shall have exactly one 'default:' No 'case:' or 'default:' shall be reachable from the previous 'case:' or 'default:' except by direct fall-through (15.2, 15.3 add-in 14.8, 15.1, 15.4, 15.5)		New	Write a new description, CLL. Using an enumerable type is a good thing. One wants the case analysis to cover all of the cases. One often wants to avoid falling through to subsequent cases. Adding a default option defeats static analysis. Providing labels marking the programmer's intentions about falling through can be an aid to static analysis.
2.15	Undefined behavior of shift operator (12.8)		add-in EWF or XYY	Added to XYY
2.16	Undefined behavior of increment (++) and decrement (--) operator (12.13)		add-in EWF	Try to deal with this in the new KOA.
2.17	~~Use of obscure language features~~ (12.10)	2	New	Write a new description, BRS, that says that guidelines for coding constructs should consider the capabilities of the review and maintenance audience as well as the writing audience, and that features that correlate with high error rates should be discouraged. Write another description, NYY, for self-modifying code that includes Java dynamic class libraries and DLLs.
2.18	Control Flow – if structure (14.9, 14.10)		New	There are two classes of languages, those that explicitly mark the end and those that don't. The former don't have this problem; the latter do. Write a new description, EOJ, that suggests writing appropriate guidelines for your language. This includes end of loop as well as if-then-else.

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6
2.19	Initialization of variables, arrays, structures and enumerated lists (9.1, 9.2, 9.3)	143, 144, 148, 71, H-6	New	Write a new description, LAV, saying that variables should not be introduced until they can be initialized with a meaningful value. (Don't do junk initialization because it defeats static analysis.) In languages that provide clear mechanism for initialization, use the clearest. For example, in Ada, use named components in aggregates. The structure of the initializer should match the structure of the initialized object. The vulnerability is that if the object's structure is changed in maintenance, the initializer simply adapts itself to the new structure and may omit values. Incomplete initializations may lead to the insertion of unexpected values, which may be wrong. When choosing a default, be explicit about it. In an enumerator list, the "=" construct should not be used unless all items are explicitly initialized (MISRA 9.3). (Steve Michell wants to do this one.) Reserve a distinct description, CCB, to discuss enumerator issues.
2.20	Loop control (13.5, 13.6, 14.6)	198, 199, 200	New	Write a new description, TEX, about not messing with the control variable of a loop.
2.21	Operator Precedence (12.1, 12.2)	204, 213	add-in FAB	We decide to write three new descriptions: operator precedence, JCW; associativity, MTW; order of evaluation, SAM.

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6
2.22	Functions – control flow/return values (16.2, 16.8, 16.10)	111, 117, 118~~, 198, 199, 200~~, 69, 116, 208	New	Write a new description, GDL, suggesting that if recursion is used, then you have to deal with issues of termination and resource exhaustion. Write a new description, NZN, about returning error status. Some languages return codes that must be checked; others raise exceptions that must be handled. Deal with tool limitations related to exception handling; exceptions may not be statically analyzable. Write another one, CSJ, to deal with passing parameters and return values. Deal with passing by reference versus value; also with passing pointers. Distinguish mutable from non-mutable entities whenever possible.
2.23	Macros (19.4, 19.7,19.8, 19.9, 19.10, 19.11)	29	New	Include in NMP
2.24	Reuse of identifiers or reserved identifiers (5.3, 5.4, 5.5, 5.6, 5.7, 20.1, 20.2)		add-in YOW	Added to YOW

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6
2.25	Overlapping or reuse of memory (18.2, 18.3, 18.4)	153, 183	New	Write a new description, AMV. Overlapping or reuse of memory provides aliasing effects that are extremely difficult to analyze. Attempt to use alternative techiques when possible. If essential to the function of the program, document it clearly and use the clearest possible approach to implementing the function. (This includes C unions, Fortran common.) Discuss the difference between discriminating and non-discriminating unions. Discuss the possibility of computing the discriminator from the undiscriminated part of the union. Deal with unchecked conversion (as in Ada) and reinterpret casting (in C++).
2.26	Restrictions on types (6.1, 6.2, 6.3, 6.5) 6.5 was done elsewhere.	148, 183	New	Write a new description, IHN, to encourage strong typing but deal with performance implications. Use enumeration types when you intend to select from a manageably small set of alternatives. Deal with issues like char being implementation-defined in C. Discuss how one should introduce names (e.g. typedefs) to document typing decisions and check them with tools.

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6
2.27	Preprocessor (14.3, 19.1)	26,30, 27, 28, 35, H-8	New	Write a new description, NMP. The use of preprocessors increases the cost of static analysis and the difficulty of human understanding. Unless the use of preprocessors is restricted to simple usage such as conditional compilation, creation of symbolic constants, and simple text insertion, issues arise concerning the analyzability of the source code and the maintainability of the generated code. Prefer language constructs to preprocessor or macro constructs whenever possible. (Consider all of the MISRA 19.x rules in this section.) Use appropriate methods to guard against multiple inclusions.
2.28	Compiler issues (1.3, 1.4~~,~~ ~~3.4~~, 5.1, 21.1)	H-10	New	Potentially for Section 7. Know how your linkage editor actually works. Understand the impact of changing your compiler switches. Select additional tooling that is appropriate. Don't ignore warnings. Analyze frequently.
2.29	Pointers (17.1, 17.2, 17.3, 17.4, 17.5, 17.6)	175	(maybe) add-in XYK or New	We decide to write a new vulnerability, Pointer Arithmetic, RVG, for 17.1 thru 17.4. Don't do 17.5. We also want to create DCM to deal with dangling references to stack frames, 17.6. XYK deals with dangling pointers.

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6
2.30	Libraries (20.3, 20.4)	16, H-7	New	Write a new item, TRJ. Calls to system functions, libraries and APIs might not be error checked. It may be necessary to perform validity checking of parameters before making the call. (However, sometimes libraries are specified to explicitly provide checking.) When writing a library for unknown users, never trust the calling programs to send the right thing. Within a system, there should be a convention established whether the caller or the called program establishes validity of the parameters. Especially when there is a distinction in privilege level. Assertions can be useful in checking preconditions.
2.31	Tool use (21.1 (good rule, but where to put it?))	H-10	New	General guidance like this will go into Section 7. Specific guidance regarding analyzability of language constructs will go into individual vulnerability descriptions.
	, 14.4, 14.5, 14.7, 20.7	Holtzmann-1	New	Write a new description, EWD, that discusses goto, structured programming, continue statement, break statement, single exit from a function. Discuss in terms of cost to analyzability and human understanding. Include setjmp and longjmp.

Number	Derived Rule w/MISRA-C:2004 Rules	JSF AV Rules	Template/New	Annotations added during Meeting #6
	20.11		New	Write a new description, REU, that discusses abnormal termination of programs, fail-soft, fail-hard, fail-safe. You need to have a strategy and select appropriate language features and library components.
		48 thru 56	New	Write a new description, NAI, on issues in selecting names. Assign this one to Steve Michell. Look at Derek's paper on the subject.
		70 thru 100 and OOTIA, 177, 178, 179, 185, 219		Consider a set of descriptions related to object-oriented programming. This is an action item for Tom Plum.
		101, 102, 103, 104, 105, 106,		Consider a description, SYM, related to templates and generics.
		120		Add this to YOW.
	20.4	H-3, 206,		Dynamic memory allocation

Table 6: Results of Mapping JSF AV Rules into Rules Derived in Part 3

Category	JSF AV Rule
Inappropriate Rules (out of scope, too language specific, style guidance, etc.)	1, 3, 4, 5, 6, 7, 14~~, 27, 28~~, 31, 32, 33, 34~~, 35~~, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, ~~48, 49, 50, 51, 52, 53, 54, 55, 56~~, 57, 58, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, ~~70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 95, 96, 97, 98, 99, 100~~, ~~101, 102, 103, 104, 105, 106,~~ 109, 110, 112~~, 116, 120, 121, 122, 123, 124, 125, 126, 128, 129, 130, 131, 132, 133, 134~~, 141, 150, 151, ~~152~~, 155, ~~159~~, 163, 169, 176, ~~177, 178, 179, 183~~, ~~185~~, ~~205~~, ~~207~~, ~~208~~, ~~216, 217,~~ 218, ~~219~~, 220, 221

Part 5: Power of 10 recommendations

Gerard Holtzmann of NASA/JPL has proposed 10 rules for developing safety-critical code. As with the other rules cited above, his rules are targeted at C, since most of the developers at JPL program in C. His rules, by his own admission, are strict. Arguably his rules will make code more dependable, but at the high cost of strict limits.

1. Rule: Restrict all code to very simple control flow constructs \u2013 do not use goto statements, setjmp or longjmp constructs, and direct or indirect recursion.

2. Rule: All loops must have a fixed upper-bound. It must be trivially possible for a checking tool to prove statically that a preset upper-bound on the number of iterations of a loop cannot be exceeded. If the loop-bound cannot be proven statically, the rule is considered violated. [Inappropriate]

3. Rule: Do not use dynamic memory allocation after initialization.

4. Rule: No function should be longer than what can be printed on a single sheet of paper in a standard reference format with one line per statement and one line per declaration. Typically, this means no more than about 60 lines of code per function. [Inappropriate]

5. Rule: The assertion density of the code should average to a minimum of two assertions per function. Assertions are used to check for anomalous conditions that should never happen in real-life executions. Assertions must always be side-effect free and should be defined as Boolean tests. When an assertion fails, an explicit recovery action must be taken, e.g., by returning an error condition to the caller of the function that executes the failing assertion. Any assertion for which a static checking tool can prove that it can never fail or never hold violates this rule. (I.e., it is not possible to satisfy the rule by adding unhelpful “assert(true)” statements.) [Inappropriate]

6. Rule: Data objects must be declared at the smallest possible level of scope.

7. Rule: The return value of non-void functions must be checked by each calling function, and the validity of parameters must be checked inside each function.

8. Rule: The use of the preprocessor must be limited to the inclusion of header files and simple macro definitions. Token pasting, variable argument lists (ellipses), and recursive macro calls are not allowed. All macros must expand into complete syntactic units. The use of conditional compilation directives is often also dubious, but cannot always be avoided. This means that there should rarely be justification for more than one or two conditional compilation directives even in large software development efforts, beyond the standard boilerplate that avoids multiple inclusion of the same header file. Each such use should be flagged by a tool-based checker and justified in the code.

9. Rule: The use of pointers should be restricted. Specifically, no more than one level of

dereferencing is allowed. Pointer dereference operations may not be hidden in macro definitions or inside typedef declarations. Function pointers are not permitted. [The good parts are captured elsewhere; the general rule is inappropriate.]

10. Rule: All code must be compiled, from the first day of development, with all compiler warnings enabled at the compiler\u2019s most pedantic setting. All code must compile with these setting without any warnings. All code must be checked daily with at least one, but preferably more than one, state-of-the-art static source code analyzer and should pass the analyses with zero warnings.

NOTE: Holzmann’s work still needs to be analyzed and incorporated.

Reference

Hatton, Les. Safer Language Subsets: an overview and a case history, MISRA C. *Information and Software Technology*. 46(7), June 2004, pp. 465-472. http://www.leshatton.org/Documents/MISRAC.pdf

Hatton, Les. EC-A Measurement Based Safer Subset of ISO C Suitable for Embedded System Development. *Information and Software Technology*, 47(3), March 2005, pp. 181-187. ISSN (online) 0950-5849 http://www.leshatton.org/Documents/ISOC_subset.pdf

Holzmann, Gerard J., *The Power of 10: Rules for Developing Safety-Critical Code*, Computer, vol. 39, no. 6, pp. 95-97, Jun., 2006

Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program, Document Number 2RDU00001 Rev C, December 2005, www.jsf.mil/downloads/documents/JSF_AV_C++_Coding_Standards_Rev_C.doc

Motor Industry Software Reliability Association (MISRA), "*MISRA-C:2004 - Guidelines for the use of the C language in critical systems*," October 2004, http://www.misra-c2.com/