C GUIDELINES COMPLIANCE AND DEVIATIONS

C GUIDELINES COMPLIANCE AND DEVIATIONS
[MISRA AND CERT CASES]

ADA-EUROPE 2018
LISBON

Maurizio Martignano
Spazio IT – Soluzioni Informatiche s.a.s
Via Manzoni 40
46030 San Giorgio di Mantova, Mantova
http://www.spazioit.com

June 2018

© 2018 Spazio IT - Soluzioni Informatiche s.a.s.
Agenda

- C compliance guidelines and deviations (intro)
- Customization examples
- MISRA and CERT compliance standardization efforts
- Analysis tools
- A viable approach to manage compliance
C compliance guidelines and deviations (intro)
MISRA C:2012 and SEI CERT C provide a set of guidelines (rules and directives – MISRA or rules and recommendations - CERT) designed to help developers in writing quality code, i.e. code that is safer, more secure, understandable and maintainable.

It is not always possible to adhere to all these guidelines and this why in several software development projects deviations and compliance levels are established in the context of so called customization or tailoring activities.
C compliance guidelines and deviations (intro)

- The customization activity can be performed very formally, adhering once again to specific and additional standards, or rather informally on a project by project base, according to the specific needs and actual limitations of the project itself.

- The customization strategies adopted by actual projects range from not allowing any deviation to accepting every documented deviation.
Customization examples
Integral types size and sign

- MISRA C 2012 – Directive 4.6 (Advisory) – “typedefs indicating size and sign should be used instead of the basic integral types”.

- This MISRA directive recommends to use the “new” C99 integral types contained in “stdint.h” instead of the “old” integral numerical types, e.g. int8_t instead of char, uint8_t instead of unsigned char, int32_t instead of int (or long depending on the hardware architecture), etc…
Integral types size and sign

June 2018

© 2018 Spazio IT - Soluzioni Informatiche s.a.s.
(Integral) types size and sign

```c
#include <stdio.h>
#include <stdint.h>
#include <windows.h>

int main(void) {
    printf("size(unsigned int) = \$zu.\n", sizeof(unsigned int));
    printf("size(int) = \$zu.\n", sizeof(int));
    // SOCKET = (unsigned int) in W32 or (unsigned int *) in W64
    printf("size(SOCKET) = \$zu.\n", sizeof(SOCKET));

    return 0;
}
```

June 2018

© 2018 Spazio IT - Soluzioni Informatiche s.a.s.
Involved rules:
- MISRA C 2012 – Directive 4.11 (Required) – “The validity of parameters passed to library functions shall be checked”.
- CERT – EXP34-C – “Do not dereference null pointers”
- CERT – MEM10-C – “Define and use a pointer validation function”

The “common sense” rule is that function parameters should be checked for their validity. In the case of pointer function parameters, a first and simple check consists in verifying they are not NULL.
(Not NULL) Pointer Function Parameters

Either the condition 'arg1=NULL' is redundant or there is possible null pointer dereference: argPtr.

Either the condition 'arg1=NULL' is redundant or there is possible null pointer dereference: argPtr.
(Not NULL) Pointer Function
Parameters

- **Design Time**
  - ASSERT type check

- **Run Time**
  - IF type check
Forbidden Conversions (and then again)

- Involved rules:

- All these rules about conversions between pointer types and/or between essential types belonging to different categories or with different sizes are sound and make sense in generic, application level pieces of code. Wherever a violation is found something suspicious is taking place and most of the times it is an error.
Forbidden Conversions (and then again)

```c
return PageRequest(conn, file, expiresPtr, &u);
}

static Ns_ReturnCode
PageRequest(Ns_CONN *conn, const char *file, const Ns_Time *expiresPtr, unsigned int afla [
    const Conn *connPtr = (const Conn *) conn;

cast from pointer to pointer [MISRA 2004 Rule 11.4, advisory] ??? 6 months ago ▼ L12
```
Forbidden Conversions (and then again)

/* We've read a chunk of the file -- all chunks (except the last) 
* will be 'BUFFER_SIZE' bytes in length; the last chunk will 
* probably be less than 'BUFFER_SIZE' bytes. */

/* Set up a pointer to walk through the buffer 4 bytes at a time */
iptr = (u_int 4 *) buffer;

for (i=0; i<length; i+=4)
{ /* Add each 4 byte piece to the checksum. */
  checksum += ntohl (*iptr);
  iptr += 4;
}
MISRA and CERT compliance standardization efforts
MISRA guidelines are divided in:

- directives: guidelines which are not defined with reference to the source code alone, but which also refer to, or impose requirements on processes and documentation;
- rules: guidelines which impose requirements on the source code and the source code only.

Rules are then divided into:

- decidable: rules that can always be assessed, verified by a (properly configured) analysis tool;
- undecidable: rules that cannot be assessed, verified by an analysis tool in every situation.
Guidelines are categorized into:

- mandatory: guidelines for which violation is never permitted;
- required: guidelines for which violations are permitted if justified by documented deviations
- advisory: guidelines that can be violated without the need of a corresponding deviation.

Resulting conclusions:

- Mandatory guidelines cannot have deviations.
- Among the required guidelines decidable rules are the less expensive to verify while undecidable rules and directives are more expensive.
- Advisory guidelines can be followed till it make sense, till it is reasonably practical to do so.
CERT compliance standardization efforts

- CERT guidelines are divided into:
  - recommendations: guidelines that are likely to improve the quality of the system;
  - rules: guidelines whose violations are likely to introduce defects which may adversely affect the safety, reliability, or security of the system.

- Each guideline contains a risk assessment based on its severity, likelihood and remediation cost; all these attributes are expressed as a number ranging from 1 to 3.
  - severity: 1 – low, 2 – medium, 3 – high
  - likelihood: 1 – unlikely, 2 – probable, 3 – likely
  - remediation cost: 1 – high, 2 – medium, 3 – low
CERT compliance standardization efforts

- The risk associated to a guideline is the product of these three attributes, which is called priority.
  - Though the product ranges from 1 to 27, only the following 10 distinct values are present in the document: 1, 2, 3, 4, 6, 8, 9, 12, 18, and 27.

- This scheme allows the definition of levels:
  - L1: guidelines with priority from 12 to 27 high severity, likely and inexpensive to fix
  - L2: guidelines with priority from 6 to 9 medium severity, probable and medium cost to fix;
  - L3: guidelines with priority from 1 to 4 low severity, unlikely and expensive to fix
Analysis tools

- Analysis tools can be divided into three broad categories:
  - “Shallow Analysers”: based on patterns matching, e.g. “PC-Lint”, “splint”, “Understand”, “Cppcheck”, ...
  - “Deep Analysers”: based on techniques like bounded model checking, semantic analysis and abstract execution, e.g. “CBMC”, “Frama-C”, “Polyspace”, ...
  - “Compiler Based Analysers”: based on the analyses performed by the compilers themselves, e.g. “Clang Static Analyzer”, “Facebook Infer”, ...

- “Shallow Analyser” are usually able to verify/assess the compliance of the majority of MISRA and CERT guidelines. In some cases, for some specific guidelines, “Deep Analysers” or manual intervention may be required.
The problem with “Deep Analysers” is that they require a lot of computing resources to perform their analyses and this might compromise their applicability to large codebases.

A recent, very interesting trend in static analysis is the adoption of “Compiler Based Analysers”; they offer several advantages, among which the most important are:

- they are the “real thing”, the very same tools used to build the software under analysis;
- they are fast and can easily analyse large codebases;
- they are easy to use, especially by developers and testers who are already accustomed to the compilation toolsets.
A practical approach
Guidelines Selection

MISRA
- Decidable Rules
- Undecidable Rules
- Directives

CERT
- L1 Guidelines
- L2 Guidelines
- L3 Guidelines
Steps

- System partitioning (based on criticality, but also on the “distance from the hardware”).

- Tools configuration (reducing “false-negatives” and “false positives”)

- (continuous) Analysis Execution

June 2018
Results Exploitation

- No “Quality-Stamps” but “Hot-Spots” in the codebase
- Training (transfer of knowledge and experience)
Results Exploitation
References

- [http://www.sonarqube.org](http://www.sonarqube.org)
- [http://sonarsrv.spazioit.com/projects](http://sonarsrv.spazioit.com/projects)
Thank you for your attention!