Ensuring Compliance to MISRA C and C++ coding standards

Software Product Assurance Workshop, September 2023

Richard Corden, Perforce Software Inc.

Agenda

- Introduction to MISRA
- Guideline Structure
- Decidability
- MISRA Compliance
- Future Plans

Introduction to MISRA

Objectives and Structure

Objectives and Structure

- Goal: “To provide world-leading, best practice guidelines for the safe and secure application of both embedded control systems and stand-alone software.”
- Work is overseen by “The MISRA Consortium Limited” (TMCL)
- Projects are delivered by Working Groups, for example:
  - Autocode
  - Safety Case
  - C
  - C++
  - Code Metrics

Working Groups

- Working Group Members formed of Domain Experts
- Members of MISRA C and C++ are:
  - Embedded Engineers
  - Language Experts
  - Tool Vendors
- Working Groups meet regularly
- Feedback on live documents available through the MISRA public forum
- New Guidelines proposed by working group members and other sources
  - AUTOSAR provided input for Guidelines in the upcoming MISRA C++ release

Guideline Structure

What makes a MISRA C/C++ Guideline?

What is a MISRA C/C++ Guideline?

- Not included in MISRA C / C++ Guidelines are:
  - Guidelines relating to Style
  - Metrics – a separate workgroup is investigating this
- Guidelines ‘subset’ the language, but not language features, for example:
  - Use `std::stoi` in preference to `atoi`
- MISRA Guidelines are the minimal set – additional domain specific guidelines can be added!

What is a MISRA C/C++ Guideline?

- MISRA C / C++ Guidelines should:
  - Be more than just “don’t make this specific mistake”
  - Be clear and unambiguous
  - Be enforceable
  - Be defensible
  - Not curtail safe use of valid language features
- Code that is compliant to the Guideline should avoid the “mistake” from taking place
- Creating such Guidelines takes time

Decidability

Scope and Decidability

Scope and Decidability

- A guideline is decidable if it can be reliably determined that code is compliant
- A C language guideline “do not modify a string literal” is not decidable
  - *`non-const`* pointer can point to string literal
  - Tracking a modification is not decidable

```
void foo (char * buffer) {
  char * chksum = “deadbeef”;
  strcpy (chksum, buffer);
}
```

Scope and Decidability

- Decidable Equivalent?
  - “Do not Assign a string literal to a pointer to *non-const*”

```
void foo (char * buffer) {
  char * chksum = “deadbeef”; /* Non-Compliant */
  strcpy (chksum, buffer);
}
```

- The C and C++ languages allow casts to remove `const`
- Requires compliance to additional Guideline: “Do not remove *const*”
- The ideal is to have Decidable guidelines, and where appropriate, “Translation Unit Scope” vs “System”.
- From C++ ‘11, conversions from string literal to `char *` were deprecated meaning that this guideline is not required for C++.

Dealing with Decidability

- Undecidable guidelines can suffer from False Positives and Negatives
- False Positives, while potentially annoying, provide a hook for developer action
- False Negatives are more serious, possible mitigations:
  - Manual code review
  - Use of multiple analysis tools
- Record in “Risk Register” what steps have been taken to mitigate the chance of False Negatives

MISRA Compliance

Achieving MISRA Compliance

What is compliance?

- It is a statement claiming that the code within a project complies with the restrictions and controls imposed by a MISRA subset (e.g. MISRA C:2023)

- A statement of compliance is a form of self-declaration
  - The organization producing the code is responsible for ensuring that it is compliant
  - Evidence needs to be produced to support a claim of compliance

- See https://misra.org.uk/app/uploads/2021/06/MISRA-Compliance-2020.pdf

Handling a Non-compliance

- Projects dealing with hardware often cannot be 100% compliant to every guideline
  - it is common for integer constant expressions to be converted to pointers to structures when accessing registers
- MISRA accepts this reality
- Deviations are use to handle a non-compliance that is unavoidable.
- MISRA Compliance should be considered as early as possible
  - Retrospective code modifications could introduce defects!

Does “compliant” mean “high quality”?

- That depends…

-
  ```
  u8  u8a;
  s8  s8a;
  u16 u16a;

  u32a = s8a * u8a; // Non-compliant
  ```
- Would a deviation be acceptable here or should the code be written in a compliant manner?
  - Both options make the code compliant, but the first would be unlikely to be consistent with high quality
- Deviations must take interaction between guidelines into account

When is a violation a “valid” deviation?

- The violation must be justifiable on strong technical grounds
  - Never just for developer convenience!
- The use of deviations must be controlled through a formal deviation process
  - Deviations are requested by a developer
  - Approved by a manager
  - Signed-off (risk accepted) by a suitable technical authority
- It is never acceptable for code to be “made compliant” by using a deviation to cover a violation which could reasonably have been avoided

Justifying a Deviation – “Reasons”

- Any deviation should be attributable to one or more of the following reasons:
  - Performance
  - Alternative build configurations
  - Access to hardware
  - Defensive coding
  - Code quality
  - Adopted code integration
  - Non-compliant adopted code

Guideline re-categorization

- MISRA allocates a category to each guideline
  - “Mandatory” – violations are never permitted
  - “Required” – violations are permitted when supported by a deviation
  - “Advisory” – violations should be avoided where practicable, but a formal deviation may not be required where violations exist
- The categories within the MISRA documents define the *minimum* enforcement level to be used for the guidelines
  - It is likely that a project will be able to raise the enforcement level for many “Required” guidelines to “Mandatory” (use of goto)
  - A project may also decide to raise “Advisory” guidelines to “Required” (or even “Mandatory”)
  - Advisory guidelines may be lowered to dis-applied (use of C++ comments in a C project)

Guideline Compliance Summary

- Minimum requirement to show project Compliance
- List of every Guideline, the category and its Compliance Level:
  - Compliant – no violations anywhere in the code
  - Deviations – violations exist, however, they are justified and formally accepted
  - Violations – violations of advisory guidelines that do not require formal justification
  - Disapplied – no checks have been made for compliance
- See MISRA Compliance 2020 for full details

Future Plans

Future Plans

- MISRA C has just published MISRA C:2023 with support for C18, including coverage of new C features such as generic selections.

- MISRA C++ will shortly publish a new release of MISRA C++, which provides guidance for the use of C++ 17.
  - Compiler support for C++'17 features is widespread and mature
  - As usage experience develops, more recent features will be included
  - Goal is to provide more regular MISRA C++ releases

- The MISRA Consortium is open to addressing languages other than C/C++ - are you interested in setting up a group and putting the work in?

Contact Details

https://misra.org.uk/

enquiries@misra.org.uk

Richard Corden

www.perforce.com

rcorden@perforce.com