WHITE PAPER

MISRA: AN OVERVIEW

You don't have to take much interest in the world of embedded software before you learn about coding standards and then become aware of MISRAC. A little further in and you become aware that there is more to MISRA than just MISRAC. This paper provides a quick overview of MISRA, coding standards and the role of static analysis tools in the development of high quality code.

PAUL BURDEN
Former member of misra c working group and co-author of misra c:2012

JILL BRITTON
Member of the MISRA C Working Group and Technical Support Manager

PRQA Programming Research

INTRODUCTION

In the early 1990s it became obvious that electronics were becoming increasingly important in cars, and that software was becoming increasingly important in electronics. With this recognition came another understanding – that software reliability was an absolute imperative for both commercial and safety reasons. Today that challenge is even greater as the amount of software within a car can now extend to as much as 100 million lines of code.

MISRA (the Motor Industry Software Reliability Association) is a consortium formed from representatives of different companies working in the automotive industry. It was set up initially with some UK government backing, to look at the challenges posed by the increasing use of software in motor vehicles and to provide guidance on how embedded software should be developed. A steady stream of documents has been published addressing various aspects of software engineering. One of the first, produced in 1994, was entitled "Development Guidelines for Vehicle Based Software", an automotive-specific interpretation of the then emerging IEC 61508 standard, "Functional Safety of Electrical / Electronic / Programmable Electronic Safety-related Systems". This document was effectively superseded with the recent publication of the ISO 26262 standard, Road Vehicles, Functional Safety.

After the funded project ended, work continued on a number of fronts but perhaps most significantly in the development of coding standards. Working groups were established and, over the years, have published a series of documents containing guidelines to address some of the problems inherent in the C and C++ languages.

C AND C++

C and C++ are by some distance the most widely used languages for embedded software development. (Recent research by VDC shows C being used by 70% of the embedded systems companies surveyed and its derivative, C++, in 42% .) C has been implemented for virtually every processor. It provides a wide range of resources and librairies, is supported by a wide range of tools, and there is a plentiful pool of developers.

> *Biocartis has used QA·C and QA·C++ during the development of IEC 62304 compliant software (safety Class C) for the Idylla™ IVD (medical) device. We apply MISRA based coding guidelines and have been very impressed by the ability of QA·C and QA·C++ to find issues that other tools miss and also the low level of false positives (noise)*
>
> Yves Willems,
> Software Engineering Manager at
> Biocartis NV

C was designed as a small, high level language to replace assembler. It has since evolved to become an application language, but its suitability for use in safety-critical environments was never a primary consideration. A C program which compiles and conforms fully to the requirements of the ISO language standard may still include code which will exhibit completely unpredictable behaviour - this is clearly unacceptable in an application such as a car braking system! The dangers can be substantially reduced by applying restrictions to the way in which the language is used and this has been the essential aim of the MISRA coding guidelines.

CODING STANDARDS

In their simplest form, coding standards are often created as a way of defining a set of consistent coding practices. Although uniformity of style can be a valuable discipline within a software project, such issues are frequently a matter of personal preference. They do not address the important attributes of software quality such as reliability, portability or maintainability. A more fundamental role of coding standards is to define a safer sub-set of the programming language by framing a set of rules which eliminate coding constructs known to be hazardous.

The C language allows the developer to do many things which are essentially incorrect. It is all too easy to write code which conforms to the requirements of the language standard but which will result in either program failure (i.e. a crash) or in undefined behaviour. Common examples might be code which results in accessing memory outside the bounds of an array or an arithmetic operation which results in integer overflow.

Clearly it is of paramount importance to identify such 'bugs'. Some may be identified by a compiler although a dedicated static analysis tool will generally be far more effective. However the primary aim of coding rules is not generally the identification of such problems but prevention. Rules which admonish the developer not to make obvious 'mistakes' are usually unhelpful and may even provoke derision! The fundamental aim of a coding rule should be to restrict use of the language so as to prevent the developer from doing things which are either intrinsically 'wrong' or potentially dangerous. For example, it is possible in the C language to declare and define functions in two different ways. The 'old' (Kernighan and Ritchie) syntax which was a feature of early versions of the language is still supported but has now been superseded by 'function prototype' syntax. The use of K&R syntax is prohibited in MISRA-C simply because it can so easily be misused and introduce bugs. Many software defects can be avoided simply by adopting some sensible restrictions on language use.

THE MISRA CODING GUIDELINES

The MISRA coding guidelines are now accepted world-wide as the benchmarks for developing safety-critical software in C and C++. They have been widely accepted because they are concise and readable and because they focus on essential issues.

Each document contains a set of coding rules, but the rules are preceded by several chapters of background information that are just as important to anyone who would like to develop robust code. The guidelines emphasize that adherence to coding rules is just one ingredient in a successful software development process. Any programming project has to be integrated into a disciplined engineering environment, with a documented development process and the use of appropriate compilers and validation tools.

> *MISRA is a very sound coding guideline that is widely adopted by developers implementing safety critical designs across a wide variety of industries and applications*
>
> Richard Burke,
> Software and Systems Manager at
> Protean Electric

MISRA C:1998

The first set of coding guidelines published by MISRA appeared in 1998. Two MISRA members, Ford and Rover, had asked PRQA to help them to create their own coding standards. This work formed the basis for what became the first edition of MISRA C: "Guidelines for the Use of the C Language in Vehicle Based Software". This version contained 127 coding rules and it made an immediate impact. Some 13 years later, it is still being used in the maintenance of many legacy systems.

MISRA C:2004

As MISRA C became widely adopted, areas were identified where it needed to be improved and clarified. A new version was published in 2004. It was structured rather differently and contained a few additional rules but preserved the essential flavour of the original version. Significantly, the modified title referred to "critical systems" rather than just "vehicle based software" - reflecting the fact that MISRA C was now widely used outside the motor industry.

MISRA C++:2008

While C remains the dominant programming language in safety critical systems, there has been a steady increase in the use of C++. In response to popular demand, a new working group was established and MISRA C++ appeared in 2008. C++ is a much more complex language than C and has a range of additional issues which require a larger set of 228 rules.

MISRA AUTOCODE

One of the growth areas in systems development is the use of modeling tools. Automatically generating code from a model is a process that is fast and flexible, especially when it comes to incorporating changes later in the product development cycle.

The application of coding guidelines in the context of automatically generated code can be a source of confusion. Rules developed for manual code development are not always appropriate for auto generated code. MISRA has therefore published additional guidelines which address the issue of how the MISRA-C:2004 rules should be applied in a code generation tool.

However, the quality of auto generated code is not the sole responsibility of the code generation tool. It may also reflect the design of the model from which the code has been generated, and so a number of documents have also been published which provide design and style guidelines in the application of modeling languages such as Simulink and Targetlink.

> *Haldex is keen to pursue "best practices" and has therefore chosen to adopt the MISRA C coding guidelines. This has not been mandated by their customers, but rather Haldex see this as a key requirement, and indeed a prerequisite to the development of high quality code for safety-critical applications.*
>
> Dudley Harrison, Chief Engineer, Trailer Systems

MISRA AC is now a few years old and far removed from the current version of the tools offered by its vendors. This document has been deprecated and no longer supported by MISRA from 1 June 2014. MISRA AC AGC will remain available and supported for legacy users of MISRA C:2004. It should be noted that the current version MISRA C:2012 integrates requirement for automatically-generated C code.

MISRA C:2012 MORE COMMONLY KNOWN AS MISRA C3

MISRA C:2012 was first published in April 2013 and marks a further step forward in the development of the MISRA C Guidelines. After 14 years of experience drawn from many thousands of users and organisations, lessons are still being learned. This, the third edition, adds some new rules, addresses some loopholes and improves the description and explanation behind existing rules including some precise guidelines on how to deviate for rules – not applicable for mandatory rules.

MISRA C:2012 extends support to the C99 version of the C language (while maintaining guidelines for C90), in addition to including a number of improvements that can reduce the cost and complexity of compliance, whilst aiding consistent, safe use of C in critical systems. Improvements, many of which have been made as a result of user feedback, include: better rationales for every guideline, identified decidability so users can better interpret the output of checking tools, greater granularity of rules to allow more precise control, a number of expanded examples and integration of MISRA AC AGC. A cross reference for ISO 26262 has also been produced.

STATIC ANALYSIS

Unfortunately, the decision to adopt coding guidelines is frequently undermined by the practical difficulties inherent in enforcing such a policy. It has been observed that many companies invest considerable effort in compiling a set of coding guidelines only to find that the document subsequently gathers dust on a shelf.

The MISRA coding guidelines recognise the fact that effective enforcement of coding rules can rarely be achieved by manual code review. Traditional code inspections are hugely time consuming and not always reliable. However, most coding rules are amenable to automatic enforcement with a tool.

Some embedded compilers provide a certain level of rule enforcement. Dedicated static analysis tools usually go much further; as well as identifying specific rule violations they may also identify other coding problems, calculate source code metrics or conduct an in-depth dataflow analysis of code to identify subtle run-time errors.

Not every MISRA rule is automatically enforceable; there are a handful of rules which address issues of documentation or which are framed so as to require an element of subjective judgement which a tool cannot provide. However the vast majority of MISRA rules can be very effectively enforced automatically.

Static code analysis is a technology which is fast, powerful, reliable and repeatable. Being a tool that can be exercised on the desktop by an individual programmer, it also avoids the confrontation and embarrassment which can easily be a feature of manual code reviews.

SUMMARY

Since publication of the first version of MISRA-C in 1998, MISRA has established a reputation as a world leader in developing coding standards for embedded software development.

C and C++ are not ideal languages for safety critical code; but it is now widely accepted that they can be used effectively to develop reliable software if a sub-set of the language is defined in a coding standard and if static analysis tools are used to provide consistent enforcement.

> *Selex ES recognizes the importance of coding standards in generating robust code, and has chosen to implement a sub-set of MISRA rules, with some company specific extensions. Although this standard was originally created for the automotive industry, the rule-set is a distillation of the work of some of the world's leading coding experts and is equally suitable for (and widely adopted across) many other industries - in particular those that are developing mission-critical and safety-critical solutions. For Selex ES the MISRA guidelines served as an ideal starting point for an application- and company-specific set of coding standards.*
>
> Ian Anderson, Head of Software Engineering at Selex ES

ABOUT MISRA

More information about MISRA can be found at http://www.misra.org.uk/

PRQA AND MISRA

PRQA's relationship with MISRA stretches back some 20 years. Major elements of both the MISRA C and MISRA C++ guidelines have been derived from our own coding standards and our technical experts remain as key members of the MISRA C and MISRA C++ working groups.

Established in 1985, PRQA is recognised throughout the industry as a pioneer in static analysis, championing automated coding standard inspection and defect detection, delivering its expertise through industry-leading software inspection and standards enforcement technology.

PRQA static analysis tools, QA·C and QA·C++, are at the forefront in delivering MISRA C and MISRA C++ compliance checking as well as a host of other valuable analysis capabilities. All contain powerful, proprietary parsing engines combined with deep accurate dataflow which deliver high fidelity language analysis and comprehension. They identify problems caused by language usage that is dangerous, overly complex, non-portable or difficult to maintain. Additionally, they provide a mechanism for coding standard enforcement.

PRQA provides Compliance Modules for the enforcement of MISRA C:1998, MISRA C:2004 and MISRA C:2012, as well as MISRA C++:2008. The effectiveness of these tools has been widely acclaimed - see the independent research conducted by TERA-Labs (a research division of the Karel de Grote University College, Antwerp, Belgium).

ABOUT THE AUTHORS

PAUL BURDEN has worked with clients around the world providing training and advice particularly in the area of coding standards enforcement. In his role as product manager for QA·C over a number of years he has gained considerable experience relating to the benefits and pitfalls of static analysis tools.

He has been a prominent member of the MISRA C Working Group since its formation more than 10 years ago.

JILL BRITTON has 30 years of embedded software experience in a variety of areas including defense, automotive, telecommunications and education. She spent 12 years working at Motorola, designing software for both network devices and automotive applications. She has also worked on Telematics for Continental Automotive.

She has in-depth knowledge of design, development and quality processes and a particular interest in coding standards and software metrics.

At PRQA, Jill leads a worldwide group of high-capability technical consultants in delivering added-value products and services to customers.

ABOUT PRQA

DETECT, ENFORCE AND MEASURE

Since 1985, PRQA has pioneered software coding governance in the automotive, aerospace, transport, finance, medical device and energy industries. Supporting both small start-ups and globally recognized brands, we provide sophisticated code analysis, robust defect detection and enforcement of both bespoke and industry coding standards through functional integrity and application security/safety.

PRQA's industry-leading solutions, QA·C, QA·C++, QA·J and QA·C# offer the most meticulous static analysis of commonly used programming languages. Innovations such as multi-threading and resource analysis (MTR) complement this with refined multi-thread inspection of code streams. Used locally or centrally deployed via the Quality Management System QA·Verify, we enable early find/fix at the desktop and on the server side complete control, visibility and history to the decision maker.

ISO 9001 and TickIT certified.

www.programmingresearch.com