ISSN XXXX XXXX © 2017 IJESC

Research Article Volume 7 Issue No.7

Using MISRA C in Software Development

Anusha .S. Pattar[1], Dr. R Nagaraja[2], Anupama .K .C[3]
M.Tech Student[1], Professor[2], Assistant Professor[3]
Department of ISE
Bangalore Institute of Technology, Bengaluru, India

Abstract:
Embedded automotive software domain demands particular requirements namely, real-time responses, precise safety constrictions, necessity of robustness for its usability. These requirements were met through multiple solutions making it more incompatible. Hence, a single standard needs to be formulated to meet the mentioned requirements. MISRA C (Motor Industry Software Reliability Association) includes a collection of guidelines for software development which is developed using C language. These guidelines aim at facilitating safety, portability, reliability and security of code in embedded systems' context developed in ISO C/C90/C99. A single literature is not available to indicate how MISRA C guidelines can be used for safety and reliability in embedded software. The objective of this paper is to emphasize the significance of using MISRA C in the development of embedded software.

Keywords: Electronic Control Units (ECUs), MISRA C, reliability, safety, and worst-case execution times (WCET).

I. INTRODUCTION

Information Technology is becoming the motivating force in various technological areas and in specific in vehicles, in embedded systems' form. Embedded software/hardware systems regulate new operations of car, help and facilitate the driver. Presently, Modern cars possess around 80 controllers where each controller has to be programmed. The driver can access numerous functions of car with the help of human machine interface which involves complex multimedia system. New functions of car are majorly developed using digital hardware and software. Embedded automotive software domain demands particular requirements namely, demand for real-time responses (engine judder or misfiring could be resulted due to the delay in transmission of ignition and injection signals), precise safety constrictions, necessity of robustness [1]. For safety-critical systems, several exclusive requirements are specified, such as the following [2]:

a) Limitation on the programming language's subset which is regarded as safe
b) Confining dataflow and control structures to accurately stated variants
c) Obeying precisely stated guidelines concerning the scope of data and functions
d) Obeying complexity measures which are predefined
e) Testability, readability and maintainability of source program

For vehicle customers, the requirements include comfort, support for navigation, increase in performance, reliable embedded systems which guarantee safety properties [3]. Until now, reliability of software and hardware systems is not sufficient in the cars. For example, in air traffic's domain, the computed reliability is greater than 10^9. This implies that mean time to failure for safety critical systems is greater than 10^9 operation hours. Such figures for reliability in cars are not known [4]. The goal of automotive systems engineer must be to distinguish among several software domains in car and provide safety, reliability and security methods for both development process and software infrastructure [5].

II. LITERATURE SURVEY

In cars, presently, individual ECUs (Electronic Control Units) comprise and handle its own information. A significant quantity of this information is interchanged using several bus systems which exist in the vehicle. Nonetheless, the existing vehicle-information is not structured as a distributed database with certain methods to accomplish, for example, data integrity. Instead, each individual ECUs store a substantial quantity of their information independently. This could direct to a situation wherein few ECUs describe the vehicle to be in motion, according to their local information, whereas other ECUs describe the vehicle to have stopped. This leads to unreliability [5]. MISRA guidelines provides reliability to prevent faults by the implementation of programming rules. It emphasizes specially on checking compliance automatically. The reason behind automatic checking is it is not of much use to define rules when those rules cannot be deployed properly. In order to ensure accurate (timing) actions of the entire system, it is necessary to be capable of computing the worst-case run time bounds for jobs of embedded software system with hard real-time restrictions. The objective of DO-178B is to offer guidelines for software development of airborne systems which accomplish their expected functions. Here, requirements related to safety are portion of the system requirements. Performance requirements are usually included in these requirements. But, explicit reference is not provided to the WCET (worst-case execution times) analysis by this [12]. MISRA guidelines presents several constraints on the subset of original C language. A few guidelines are identified which have direct impact on WCET analysability. The inferences of these guidelines are of substantial significance for WCET analysis.

III. USING MISRA GUIDELINES FOR SOFTWARE DEVELOPMENT

This section presents how MISRA guidelines can used for reliability and safety in embedded software development.

i) **Reliability:** Reliability of software is an important quality attribute in automotive domain. Reliability and safety

concerns are significant for every driving related function, functions related to passenger safety and engine control functions. Coding standards have been used by developers which guides them to a universal style and to stop them from making use of constructions which are identified as potentially problematic. Examples include Sun's internal Java coding guidelines which helps the beginners to understand the code easily, MISRA C which is targeted particularly towards safety of C programs in critical systems [8]. Guidelines in these are generally established on expert views and could be directed towards many goals like maintainability, reliability or portability. Cathal Boogerd et al. presents a paper which focuses on reliability, that is, to prevent faults by the implementation of programming rules [9]. In this circumstance MISRA is of specific interest since it emphasizes specially on checking compliance automatically. The reason behind automatic checking is it is not of much use to define rules when those rules cannot be deployed properly. The empirical assessment of relation between actual faults and violation of standard coding rules is presented in this literature. They define violation as indication that a source code location does not conform to any rules of the standard and fault as the issue available in tracking system for issues which might be related to many source code locations. Cathal Boogerd et al. proposes two approaches to enumerate relation between faults and violations and also explain their requirements associated with richness of data set for input [9]. The first is a more high-level approach, studying the co-evolution of violations and faults over time, testing for the presence of a *correlation*. The second approach looks at individual violations in more detail, tracking them over time and seeing how often they correctly signal actual problem locations, giving us a *true positive rate* for every rule. They also offer observed data achieved by employing both approaches to an embedded software project in an industry and the coding standard MISRA C 2004. This data comprises of relationship between faults and rule violations along with gauged rates of true positive for each rule. Cathal Boogerd et al. concludes by emphasizing that it is vital to choose precise and appropriate rules. Choice of rules which would mostly account for rise in reliability increases the advantage of conformance while decreasing essential effort. Furthermore, empirical proof provides matter for arguments of supporters of coding standards which makes adoption of coding standard easier in organization. Nonetheless, relationships and rates of true positive measured in this experiment might differ project to project. In order to grow more confidence in their results and to scrutinize if they could differentiate a reliable subgroup of MISRA rules that are positively associated to actual faults, they aim to repeat this research for multiple projects.

ii) Safety:

Safety is of significant importance while developing embedded software. Reliability and safety concerns are significant for every driving related functions, functions related to passenger safety and engine control functions. In order to ensure accurate (timing) actions of the entire system, it is necessary to be capable of computing the worst-case run time bounds for jobs of embedded software system with hard real-time restrictions. Any method to improve the (static) time predictableness of embedded software system hold great interest particularly because of the ever-increasing intricacy of such kinds of software systems. Gernot Gebhard et al. presents a paper which studies the present coding guidelines and proposals, namely MISRA C and scrutinize if they make analysis of static timing simpler [13]. Numerous coding guidelines had been emerged to help programmers to write programs that conform to principles of safety-critical software. The main objective is to create programs that do not comprise of errors which would lead to serious failure and hence harming equipment or individuals. MISRA, in 1998, presented MISRA C [6]. These guidelines were aimed for automotive embedded systems which were developed using C programming language. An upgraded version of these guidelines was released in 2004 [8]. These standard guidelines are broadly accepted in present days in different safety-critical domains, namely defense systems or avionics. Wenzel et al. considers that only MISRA C comprises coding guidelines that can influence software predictability amongst standards such as MISRA C, ARINC 653 and DO-178B [12]. Following is the list of rules which would influence time predictability making use of static WCET (worst-case execution times) analysis at binary-level (e.g., with the aiT tool).

a) Rule 13.4 (required): ***The controlling expression of a for statement shall not contain any objects of floating type*****.** By preventing floating point values in looping conditions, loop analysis is facilitated to identify loop bounds automatically.

b) Rule 13.6 (required): ***Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop*****.** This rule supports the using counter-based loops and prevents the employment of complicated logic for updating the loop counter. This permits for an easier and simpler detection of loop bound.

c) Rule 14.1 (required): ***There shall be no unreachable code*****.** Few tools can identify that some portion of the code is unreachable. Yet, analysis of static timing calculates an over-estimation of possible control-flow. Because of this, analysis could presume few execution paths which are infeasible in actual software execution. Thus, removing unreachable portion from codebase directs to lesser sources of such inexactness.

d) Rule 14.4 (required): ***The goto statement shall not be used*****.** Using goto statements essentially does not produce problems for timing analysis at binary-level. But, using goto statements could possibly present loops which are irreducible, into the code binary. A recognised method is not available to identify loop bounds automatically for this type of loops. Subsequently, manual annotations are required always.

e) Rule 14.5 (required): ***The continue statement shall not be used*****.** Unstructured loops could be resulted by not obeying to this rule. Nonetheless, continue statement cannot direct to loops that are irreducible. A loop compromising of continue statement could be converted into a semantically alike loop by using if-then-else statements. Thus, the only objective of this guideline is to impose a specific coding style.

f) Rule 16.1 (required): ***Functions shall not be defined with a variable number of arguments*****.** Functions with varying list of arguments innately direct to loops which are data dependent, reiterating the list of arguments. These loops are difficult to automatically bound.

g) Rule 16.2 (required): ***Functions shall not call themselves, either directly or indirectly*****.** Similar to usage of goto statements, using recursive function call could lead to loops that are irreducible. Hence, a comparable effect would apply on software predictability as described for goto statement.

h) Rule 20.4 (required): ***Dynamic heap memory allocation shall not be used*****.** Dynamic allocation of memory directs to memory addresses that are statically unknown. This would lead to an over-approximation in the existence of many memory areas or caches with distinct timings.

i) Rule 20.7 (required): ***The setjmp macro and the longjmp function shall not be used*****.** In agreement to the

guidelines described above (14.4 and 16.2), using setjmp and longjmp functions would lead to loops that are irreducible. Therefore, comparable time predictability issues would arise. It is very difficult or impossible, yet for the limited group of real-time code conforming to their regular coding rules, to determine exact WCETs. A static analysis of WCET should deal with many challenges to be fruitful. Gernot Gebhard et al. distinguishes two different groups of challenges. One group represents the challenges that have to be encountered for making computation of WCET bound practical are termed tier-one. The other group represents the challenges that are involved in preserving WCET bounds as strict as possible. Gernot Gebhard et al. concludes by saying that standards such as MISRA C partially tackle tier-one challenges faced while carrying out WCET analysis. Automatic generation of code is being used increasingly for software development in automotive industries. This is for dealing with present days' growing requirements regarding time required for development of ECU and cost reduction. On the other hand, automatic generation of code is scarcely employed in present days in safety-critical systems' development. Grounds for this include particular requirements needed on code and lack of experience in safety-critical software development. Particular requirements on the quality of code are described in MISRA C standard. Michael Beine et al. presents a paper that deals with employing automatic generation of code for safety-critical systems' development [2]. Their paper employs TargetLink which is a production code generator of dSPACE as an illustration. The code generated using TargetLink conforms to majority of 127 MISRA guidelines. MISRA specifically allows divergences from standard if those are documented and technically acceptable. Such a conformity document describing divergences from MISRA standard has been presented by dSPACE [7]. Michael Beine et al. concludes by saying that automatic generation of code offers numerous advantages for users and could be employed for safety-critical systems' development. It is appropriate for software which has to conform to safety standards namely IEC 61508 SIL3 when implanted into sufficient development process. There are many particular requirements enforced on code generator associated with quality and technology which need to be met. TargetLink which is a production code generator of dSPACE conforms to these requirements and henceforth it is an appropriate code generator for safety-critical systems' development. Günter Heiner et al. recommends a subset of ANSI C [10] during designing of time-triggered applications which can be imposed in numerous ways of diverse strictness, for example by making use of a particular coding guideline or by making use of a particular pre-processor or analysis tool, in their paper [11] which presents a new computer architecture, the Time-Triggered Architecture, for distributed fault-tolerant embedded real-time systems.

IV. CONCLUSION

Information Technology is becoming the motivating force in various technological areas and in specific in vehicles, in embedded systems' form. The paper emphasizes on few requirements of the automotive software such as safety, reliability and so on. It is shown that MISRA C can be used for meeting the mentioned requirements.

V. REFERENCES

[1].A. Fleischmann, J. Hartmann, C. Pfaller, M. Rappl, S. Rittmann, D. Wild, "Concretization and Formalization of Requirements for Automotive Embedded Software Systems Development", In: The Tenth Australian Workshop on Requirements Engineering (AWRE), Melbourne, Australia, K. Cox, J. L. Cybulski et. Al (ed.), 2005, 60 - 65.

[2].Michael Beine, Rainer Otterbach, Michael Jungmann, "Development of Safety-Critical Software Using Automatic Code Generation", SAE Technical Paper 2004-01-0708, 2004.

[3].Fran_coise Simonot-Lion, "In car embedded electronic architectures: how to ensure their safety", 5th IFAC International Conference on Fieldbus Systems and their Applications - FeT'2003, 2003, Aveiro/Portugal, pp.1-8, 2003.

[4].Manfred Broy, "Automotive Software and Systems Engineering", MEMOCODE '05: Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design, pp. 143-149, 2005.

[5].Manfred Broy, Ingolf H. Kru¨ger, Alexander Pretschner, and Christian Salzmann, "Engineering Automotive Software", Proceedings of the IEEE, 95(2):356-373, 2007.

[6].MISRA Guidelines for the Use of the C Language in Vehicle Based Software, April 1998.

[7].Thomas Thomsen, "Integration of International Standards for Production Code Generation", SAE Technical Paper 2003-01-0855, 2003.

[8].The Motor Industry Software Reliability Association (MISRA). Guidelines for the use of the C language in critical systems, October 2004.

[9].Cathal Boogerd, Leon Moonen, "Assessing the Value of Coding Standards: An Empirical Study", Software Maintenance, 2008. ICSM 2008, IEEE International Conference on, pp. 277-286, 2008.

[10].The Motor Industry Software Reliability Association (MISRA): "Development Guidelines for Vehicle Based Software: C Sub-set", Version 0.9, August 1997.

[11].Günter Heiner, Thomas Thurner, "Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems", Fault-Tolerant Computing, 1998. Digest of Papers. 28th Annual International Symposium on.

[12].Ingomar Wenzel, Raimund Kirner, Martin Schlager, Bernhard Rieder, and Bernhard Huber. "Impact of dependable software development guidelines on timing analysis", In Proceedings of the 2005 IEEE Eurocon Conference, pages 575–578, Belgrad, Serbia and Montenegro, 2005. IEEE Computer Society.

[13].Gernot Gebhard, Christoph Cullmann, and Reinhold Heckmann, "Software Structure and WCET Predictability", Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES 2011), OpenAccess Series in Informatics (OASIcs) Series, vol. 18, Schloss Dagstuhl -- Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 1--10.