# A Static Robustness Grid Using MISRA C2 Language Rules

Mohammad Abdallah
School of Engineering and Computing Sciences
Durham University
Durham, UK
m.m.a.abdallah@dur.ac.uk

Malcolm Munro
School of Engineering and Computing Sciences
Durham University
Durham, UK
malcolm.munro@dur.ac.uk

Keith Gallagher
Department of Computer Sciences
Florida Institute of Technology
Florida, USA
kgallagher@fit.edu

***Abstract*—Program robustness is the ability of software to behave correctly under stress. Measuring program robustness allows programmers to find the program's vulnerable points, repair them, and avoid similar mistakes in the future. In this paper, a *Robustness Grid* will be introduced as a program robustness measuring technique. A Robustness Grid is a table that contains rules classified into categories, with respect to a program's function names and calculates robustness degree. The Motor Industry Software Reliability Association (MISRA) rules will be used as the basis for the robustness measurement mechanism. In the Robustness Grid, for every MISRA rule a score will be given to a function every time it satisfies or breaches a rule. The Robustness Grid shows how much each part of the program is robust, and assists developers to measure and evaluate robustness degree for each part of a program.**

***Keywords-Robustness; Robustness Grid; MISRA C2.***

## I. INTRODUCTION

Robustness is required in critical programs where failures could cause problems [1]. Robustness is an important factor in any program development process. The IEEE defines robustness as "*The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions*" [2].

In this definition, there are three main aspects; the correct program response, the input data, and system environment. Program response means that the system should respond rationally [3], but not necessarily correctly. It should not fail to reply or react illogically. The input data is one of the factors that affect the robustness of the program. A robust program can continue to operate correctly despite the introduction of invalid input [4].

The environment where the program is run is contained in hardware, other software systems, and the humans that run the program. These factors also affect the program robustness. It is this aspect of robustness that this study is concerned with.

Static measures of software robustness complement robustness testing. Robustness testing is a "*testing methodology to detect vulnerabilities of a component under unexpected inputs or in a stressful environment.*" [5]

The objective is to evaluate the robustness features of imperative programs from the perspective of programmers and maintainers. Thus it will give an assessment of the program vulnerabilities, in order to help improve and certify the robustness of existing programs.

The Robustness Grid is developed as a measurement tool and is a numeric representation of the robustness degree for each function, and for the program in total. The Robustness Grid certifies C program robustness through applications of MISRA C2 guidelines.

There are different standards that the programmers are advised to follow during writing a C program to produce a robust program. However, these standards are not widely used to measure the program robustness after the program has been written.

This study will contribute a Robustness Grid using a number of robust features. The MISRA C2 language rules will be used as a measurement of the robustness features. The Robustness Grid will provide the robustness degree for the program, and each function it includes, as a numeric value. Thus the Robustness Degree will show the degree of satisfaction that a program has according standards of robustness.

In section 2, MISRA C language rules are presented. Section 3 overviews the existing research in Robustness Grid technique. In Section 4 the Robustness Grid Calculations concepts are listed. Section 5 presents the related work in Robustness measurement. Finally, in Conclusions future research is highlighted.

## II. MISRA C2

The Motor Industry Software Reliability Association (MISRA) has published a standard set of rules for C and C++ "*to provide assistance to the automotive industry in the application and creation within vehicle systems of safe and reliable software*" [6]. MISRA C 1998 rules ("MISRA C1") where published in 1998 and were followed by technical clarification document in 2000. In 2004, MISRA published a second version of MISRA C rules (MISRA C2) to address some technical and logical problems, and for further technical clarification. In MISRA C2 the rules are rephrased to be more sensible, accurate and comprehensive.

MISRA C2 rules are classified into two types: Required (122 rules) and Advisory (20 rules). Required rules are obligatory and must be followed by developers to create safe programs. Advisory rules are necessary but not as important as the Required rules; however a developer should follow the advisories in order to build a safe program. In addition, MISRA C2 has 21 categories that consider different programming processes, coding styles, and programming syntax.

The MISRA categories cover all C language common programming issues. The MISRA categories start with Environment category, which describes the optimum environment for C programs. Then Language extensions category, where it has headlines for writing comments through programming. Documentation category contains general rules for documentation process.

The syntax format concerns, problems and advice is covered and discussed in the rest of the categories. An example of a MISRA C2 rule is rule 8.1:

> *Rule 8.1 (required) Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call.* [4]

"X.y" is the MISRA rule numbering method and means this is rule 1 ("y") in category 8 ("X") (Declarations and definitions). "*required*" means it the rule is an obligatory rule.

## III. ROBUSTNESS GRID

Measuring software robustness needs to examine features in order to produce a relative scale that calculates the robustness degree for functions, and the entire program.

### A. Robustness Features and Robustness Degree

Before discussing the Robustness Grid, some terms should be clarified: Robustness Features and Robustness Degree.

*Robustness Features* are characteristics that affect software robustness, such as code syntax [7]. Robustness Features in this study are divided into two groups depending on their source. Robustness Language Features certify the robustness degree of code syntax and coding style. Second, User Functional Requirements features certify the robustness degree of the service that program provides, how it reacts to input, and how the system responds [8].

*Robustness Degree* is a scale of a program robustness features satisfaction, expressed as a percentage.

MISRA C rules are divided to several Categories as described in following section. These categories will be used to create the Robustness Grid.

### B. Robustness Grid

The *Robustness Grid* is a table showing the robustness degree of every function in a program and for the entire program. Then the robustness features satisfaction percentage will be calculated cumulatively in each category, function, and whole program. The values highlight the vulnerable points (low percentage score) of the functions and program. *TABLE II* shows an example of the Robustness Grid. Each category in the Robustness Grid is independent, so a function could score a high marks in one category and score low marks in another.

The Robustness Grid has two parts: the *static part* which contains the MISRA C2 rules; here, there is no need to understand the code functionality because only the program code will be certified. The second part is the *dynamic part*, which contains User Functional Requirements. In this paper, only the static part will be discussed.

#### 1) *Rules selection method and conditions:*

In this study, some assumptions and conditions are applied to programs to be certified by the Robustness Grid:

1. The program must be compileable by a compiler that satisfies the MISRA C2 environment rules.
2. Programs should satisfy MISRA C2 rules number 1.1 and 14.2 which means the program must satisfy the ISO Standards [9].

The total number of MISRA C2 rules, after applying Robustness Grid assumptions and conditions is 100 in 6 Categories.

#### 2) *Rule categorization method:*

The Robustness Grid (TABLE II) is a table that classifies MISRA C2 rules into 6 different Categories; each Category has a set of related rules:

TABLE I. ROBUSTNESS CATEGORY CONSTRUCTIONS

| Category | Constructs |
|---|---|
| 0 | The rules that considers type definition, arithmetic statements. |
| 1 | Rules that consider control statements (if, for, while …etc). |
| 2 | The rules that consider function structure. |
| 3 | The rules that consider arrays, pointers, and data structure (union, struct, enum …). |
| 4 | The rule that consider header files and the pre-processor |
| 5 | All MISRA C2 advisory rules. |

If a rule is in more than one Category, it will be classified under the highest Category. If a single line of code is considered by more than one Category, it will be certified against each Category, individually.

## IV. ROBUSTNESS GRID CALCULATIONS

Certifying program robustness using the Robustness Grid uses the following procedure:

1. The program must be able to be compiled by the *gcc* compiler.
2. Pre-processor code lines are considered as part of the function main, unless it related to a particular function.

3. Rules which are not applicable for all functions in a program are removed from Robustness Grid in order to save space.

After the rules have been selected, and program eligibility is satisfied, the Robustness Grid is built for the program. In the Robustness Grid, the calculations that measure the Robustness Degree for the functions and for the program is novel and introduced here for the first time.

The Robustness Grid building process is as follows:

1. Each statement in the program is assessed against all the selected MISRA C2 rules.
2. All selected rules will be put in their categories depending on the categorisation method defined above.
3. Each rule has the applied status next to it, showing whether it is satisfied (+), violated (-), or not applicable (0).
4. Program Statements will be grouped by their function.
5. For each function the status of all rules is listed.
6. The Robustness Grid calculations are made for each function (FACS), category (ACD), and for the entire program (WPCS).

The *SwapAdd.c* program is a simple example program that will be used to illustrate how the Robustness Grid is applied. The *SwapAdd.c* program, shown in Fig 1, is a C program with three functions, *main, swap, and incr*. The *swap* function exchanges two pointers and *incr* function increments its first parameter by the value in its second parameter, and *main* calls both functions.

In program SwapAdd.c, *incr*, *swap*, and *main* are the program functions. In the Robustness Grid the numbers under each function are the rule states; a positive number (+n) means the rule has been satisfied *n* times in the function. Negative numbers (-n) means the rule has been broken *n* times in the function. Zero means the rule is not applicable to the code.

The robustness degree can be calculated as follows:

1. **Function Category Satisfaction (FCS):** For Category n the number of times a rule has been satisfies divided by the number of times the rule has been applied, expressed as percentage.
2. **Program All Categories Satisfaction (PACS):** For Category n, a count of all the times a rule has been satisfied for all the program's functions divided by the number of times the rule has been applied in all program functions, expressed as percentage.
3. All Categories (between 0 and n) **Accumulative Robustness Degree (ACD):** Number of times rules are satisfied in categories (0 - n) divided by number of times rules are applied in categories (0 - n), expressed as percentage.
4. **Function All Categories Satisfaction (FACS):** Number of times rules are satisfied in all categories divided by number of times rules are applied in all categories, expressed as percentage.
5. **Whole Program Categories Satisfaction (WPCS):** For all program functions: Count of all times rules been satisfied divided by all times that rules been applicable as a percentage.

```
#include <stdio.h>
#define LAST 10
void incr(int *num, int i);
void swap(int *a, int *b);
int main(){
  int i, sum = 0, *a = 12,*b = 13;
  for ( i = 1; i <= LAST; i++ ) {
     incr(&sum, i);}
  printf("sum = %d\n", sum);
  swap (&a,&b);
  return 0;
}
void incr(int *num, int i) {
   *num = *num + i;
}
void swap(int *a, int *b)  {
   int temp= *a;
   *a= *b;
   *b= temp;
   printf ("pointer a is:%d\n",*a);
   printf ("pointer b is:%d\n",*b);}
```

Figure 1. SwapAdd.c Program

The result of analysis as shown is TABLE II shows that the *SwapAdd.c* program satisfied the robustness features by 75.5%. To improve the robustness of the program the category 5 rules should be examined because they have the smallest PACS ratings. It also shows that *swap* should be examined because has the smallest FCS value.

This static Robustness Grid is still produced manually, which is a limitation of this study. The automation for the Grid will be done by using the semantic part of C language.

## V. RELATED WORK

Critical programs must be robust to avoid the problems that could be caused by failures [10]. The C Language standards were introduced to avoid the code misinterpretation, misuse, or misunderstanding. The IEEE has the ISO/IEC 9899:1999 standard [9], which is used later by MISRA to produce MISRA C1 and C2. This in turn led to Jones producing "The New C Standard: An Economic and Cultural Commentary" [10]. The LDRA Company uses MISRA C rules in addition to 800 rules that it created to assess programs [11]. Other C standards such as "C programming language Coding guideline" [12] are less frequently used.

Measuring the application of language standard to a program is one program robustness measurement technique. Several techniques have been tried to measure program robustness. Software measurement means estimates the cost, determine the quality, or predict the maintainability [13]. Arup and Daniel [14] presented features such as portability to evaluate some existing benchmarks of Unix systems. As a result they built a hierarchy structured benchmark to identify robustness issues that have not been detected before. Behdis and Shokat [15] introduced a theoretical foundation for a robust matrices that reduce the uncertainty in distributed system. Arne et al. [16] used some robustness criteria such as input date rate, and CPU clock rate to create a multi-dimensional robustness matrices and use them to measure the robustness of a system.

A Robustness Hierarchy is a relative scale to find the robustness characteristics that needs to be added to programs. A Robustness Hierarchy is a technique used to build a robust program. The Hierarchy starts with non-robust program as first step then adds robust features before reaching a robust program in the highest level of the Hierarchy [7].

All the previous software measurement techniques do not give the developer a fully detailed set of measurements. Nor do they specify the parts of the program that need to be modified to raise its quality. Thus the focus of this study is to give the programmer a full description for all the robustness features and the degrees to which they are satisfied.

The Robustness Grid allows the developer to specify the code lines that need to be modified to improve the program Robustness Degree.

## VI. CONCLUSION AND FUTURE WORK

A Robustness Grid has been defined and it has been shown how it can work as an assessment tool. This means that every function in a program can be certified using MISRA C2 rules through the calculation of a robustness degree.

The Robustness Degree show the MISRA C2 rules that have been followed and satisfied by the program and the rules that have been violated. The Robustness Degree gives an indication as to where the developer or maintainer should do some code changes to improve the robustness of the program.

The calculation of the Robustness Degree can be considered as simplistic in that is based on percentages of rules that are passed or failed. It does not fall into the trap of allowing positive and negative values to cancel each other out. All rules are treated with the same weight. It is clear that for any particular program this is not necessarily so. In the future work, a dynamic robustness features will be introduced to make the measurement more accurate and reliable by giving weights to the important statements in the program. Thus in the Robustness Grid, each static rule will be weighted by the Dynamic rules to highlight the different level of importance of the static rules.

## REFERECES

[1] G.M. Weinberg, Kill That Code!, Infosystems, 1983, pp. 48-49.

[2] IEEE, IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, IEEE Computer Soc, 1990.

[3] S.D. Gribble, Robustness in complex systems, Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, 2001, pp. 21-26.

[4] L.L. Pullum, Software fault tolerance techniques and implementation, Artech House, Inc., 2001.

[5] L. Bin, L. Xuandong, L. Zhiming, M. Charles, and S. Volker, Robustness testing for software components, Elsevier North-Holland, Inc., 2009, pp. 879-897.

[6] M.I.S.R. Association, MISRA website, last access <retrieved: 7, 2011>.

[7] M. Abdallah, M. Munro, and K. Gallagher, Certifying software robustness using program slicing, 2010 IEEE International Conference on Software Maintenance, Timisoara, Romania, 2010, pp. 1-2.

[8] I. Sommerville, Software Engineering, Addison-Wesley, 2006.

[9] ISO/IEC, International Standard ISO/IEC 9899, International Organaization for Standardization, 1999.

[10] D.M. Jones, The New C Standard: A Cultural and Economic Commentary, Addison-Wesley Professional, 2003.

[11] LDRA, LDRA Test Suite, last access <retrieved: 7, 2011>.

[12] E. Laroche, C programming language coding guidelines, last access <retrieved: 7, 2011>.

[13] N.E. Fenton, and S.L. Pfleeger, Software Metrics, A Rigorous and Practical Approach, PWS Publishing Company, 1997.

[14] A. Mukherjee, and D.P. Siewiorek, Measuring Software Dependability by Robustness Benchmarking. IEEE Transactions of Software Engineering 23 (1994) 94-148.

[15] B. Eslamnour, and S. Ali, Measuring robustness of computing systems. Simulation Modelling Practice and Theory 17 (2009) 1457-1467.

[16] A. Hamann, R. Racu, and R. Ernst, Methods for multi-dimensional robustness optimization in complex embedded systems, Proceedings of the 7th ACM &amp; IEEE international conference on Embedded software, ACM, Salzburg, Austria, 2007, pp. 104-113.

TABLE II. SWAPADD.C ROBUSTNESS DEGREE

| Category | C2 Rules | incr | FCS% | swap | FCS% | main | FCS% | PACS% |
|---|---|---|---|---|---|---|---|---|
| **Category 0** | 4.1& 7.1 | 0 | 3/3 = 100% | +2 | 5/7 = 71.4% | +1 | 6/8 = 75% | 14/18 = 77.8% |
| | 4.2 | +2 | | +2 | | +4 | | |
| | 5.1 | +1 | | +1 | | +1 | | |
| | 5.2 | 0 | | -2 | | -2 | | |
| **Category 1** | 12.2 | +1 | 4/5 = 80% | 0 | 5/5 = 100% | -1 | 8/11 = 72.7% | 17/21 = 81% |
| | 13.1 | +1 | | +3 | | +4 | | |
| | 13.4 | 0 | | 0 | | +1 | | |
| | 13.5 | 0 | | 0 | | -2 | | |
| | 13.6 | 0 | | 0 | | +1 | | |
| | 14.7 | +1 | | +1 | | +1 | | |
| | 17.1 | -1 | | 0 | | 0 | | |
| | 17.5 | +1 | | +1 | | +1 | | |
| **ACD (0 – 1)** | | | 87.5% | | 83.3% | | 73.7% | 79.5% |
| **Category 2** | 8.1 | +1 | 10/12 = 83.3% | +1 | 8/13 = 61.5% | 0 | 9/12 = 75% | 27/37 = 73% |
| | 8.2 | +1 | | +1 | | +1 | | |
| | 8.3 | +1 | | +1 | | +1 | | |
| | 8.6 | +1 | | +1 | | +1 | | |
| | 8.11 | -1 | | -1 | | -2 | | |
| | 14.8 | 0 | | 0 | | +1 | | |
| | 16.1 | +1 | | +1/-2 | | +3 | | |
| | 16.2 | +1 | | 0 | | 0 | | |
| | 16.3 | +2 | | +2 | | 0 | | |
| | 16.4 | +1/-1 | | -2 | | 0 | | |
| | 16.5 | 0 | | 0 | | -1 | | |
| | 16.8 | 0 | | 0 | | +1 | | |
| | 16.9 | +1 | | +1 | | +1 | | |
| **ACD (0 – 2)** | | | 85% | | 72% | | 74.2% | 76.3% |
| **Category 3** | 16.7 | +1 | 100% | +2 | 100% | 0 | 0 | 100% |
| **ACD (0 – 3)** | | | 85.7% | | 74.1% | | 74.2% | 77.2% |
| **Category 4** | 19.6 | 0 | 2/2 = 100% | 0 | 2/2 = 100% | +1 | 4/4 = 100% | 8/8 = 100% |
| | 20.1 | +1 | | +1 | | +1 | | |
| | 20.2 | +1 | | +1 | | +1 | | |
| | 20.9 | 0 | | 0 | | +1 | | |
| **ACD (0 – 4)** | | | 87 | | 75.9 | | 77.1 | 79.3 |
| **Category 5** | 5.7 | -1 | 1/2 = 50% | -2 | 1/3 = 33.3% | -3 | 3/6 = 50% | 5/11 = 45.5% |
| | 19.1 | 0 | | 0 | | +1 | | |
| | 19.2 | 0 | | 0 | | +1 | | |
| | 19.7 | +1 | | +1 | | +1 | | |
| **FACS** | | | 84% | | 71.9% | | 73.2% | WPCS **75.5%** |