MISRA C++

1 tot 2 jaar om alle aspecten van het document te implementeren...

De Motor Industry Software Reliability Association (MISRA) heeft de richtlijnen gepubliceerd voor het gebruik van C++ in kritische systemen. Het rapport specificeert een deelverzameling van C++, aanbevelingen voor het gebruik daarvan, en ook algemene regels die de kans op softwarefouten verkleinen. Het MISRA C++ document moet een vergelijkbare invloed krijgen als het bekende MISRA C boek van 1998, zo hoopt de organisatie. Een voorbeeld van een MISRA C++ regel: "Rule 9-5-1 (Required) Unions shall not be used".

HANS VAN THIEL

De belangrijkste programmeertaal voor embedded systemen is C en veel ontwikkelaars zullen bekend zijn met het MISRA C boek. Dit document is in 1998 gepubliceerd door de Motor Industry Software Reliability Association en bevat richtlijnen voor het schrijven van betrouwbare software in C.

De MISRA organisatie is opgericht door een aantal autofabrikanten en toeleveranciers en heeft inmiddels zo'n 30 rapporten en gidsen uitgebracht. Die documenten specificeren allemaal een aspect van software in voertuigen en proberen een verbinding te leggen tussen automobielengineering en software-engineering.

De achterliggende filosofie is dat veiligheid, robuustheid en betrouwbaarheid van begin af aan in de software moeten zijn ingebouwd, en dat die eigenschappen ook in de code te zien moeten zijn. De aanwezigheid van 'dead code', bijvoorbeeld, is in strijd met dat laatste principe. (MISRA C++ verbiedt overbodige code dan ook expliciet in regel 0-1-9).

Het MISRA C boek behandelt ook een aantal aspecten van C die implementatieafhankelijk zijn of ongedefinieerd in de ISO (International Standards Organization) standaard. De grootte van een int type in C kan bijvoorbeeld 16, 32 of 64 bit zijn (of nog iets anders) en MISRA C (en ook MISRA C++) beveelt aan om alleen met naar grootte gedefinieerde basistypen te werken.

Het objectgeoriënteerde C++ is, op een paar kleine verschillen na, een superset van C. Met C++ kan dus op een hoog abstractieniveau worden gewerkt, met behoud van de afstemming en control op C-niveau. Tevens zijn in C++ een aantal standaardfuncties en -bibliotheken van C vervangen door betere en gemakkelijkere equivalenten.

De oude C varianten kunnen echter ook in C++ worden gebruikt (al zou dat in strijd zijn met MISRA C++ regel 18-0-1, die dat expliciet verbiedt). Maar de overgang van C naar C++ kan geleidelijk verlopen, wat uiteraard een groot voordeel is. C++ wordt dan ook meer en meer ook voor (kritische) embedded toepassingen gebruikt, zoals in de Joint Strike Fighter (JSF). Dit toenemende gebruik van C++ was in 1995 de aanleiding voor MISRA om aan het document te gaan werken dat nu is verschenen: "MISRA C++ 2008: Guidelines for the use of the C++ language in critical systems".

Regels

De kern van het 220 bladzijden tellende boek wordt gevormd door regels zoals in de afgebeelde listing 1. Die zijn nog eens samengevat in appendix A. Appendix B is een opsomming van mogelijke problemen (kwetsbaarheden) rond het gebruik van C++ en Appendix C is een verklarende woordenlijst. Het boek begint met een inleiding en een toelichting op de achtergronden en het gebruik.

Elke MISRA C++ regel heeft een samengesteld nummer aa-bb-cc. De eerste twee delen verwijzen naar de corresponderende sectie in de ISO/IEC 14882:2003 C++ standaard. Als het eerste getal (aa) gelijk is aan 0, dan is de regel niet specifiek voor C++. Als bb gelijk is aan 0, dan wordt er niet verwezen naar een specifieke taalconstructie. Het derde getal (cc) is een volgnummer in het MISRA document.

Een regel kan '*Required*', '*Advisory*' of '*Document*' zijn. 'Document' is de strengste variant; afwijkingen zijn niet toegestaan. Dat geldt weliswaar ook voor 'Required', maar van dergelijke regels kan worden afgeweken, mits die afwijking duidelijk gemotiveerd is.

Zo is, bijvoorbeeld, regel 9-5-1 die het gebruik van het 'union' type 'Required' verbiedt , maar als de programmeur die taalconstructie toch wil gebruiken, dan kan dat, mits de afwijking beargumenteerd is.

In een 'specific deviation request' moeten dan de details van de afwijking zijn aangegeven, de mogelijke gevolgen van de afwijking, de rechtvaardiging van de afwijking en een demonstratie van hoe de veiligheid is verzekerd. In het geval van de 'union' zijn er dan ook nog regels die het gebruik nader reguleren, als er dan toch van de hoofdregel wordt afgeweken. MISRA C++ kan dus overlappende of 'redundant' regels bevatten, zoals de auteurs ook expliciet aangeven.

Een regel kan vervolgens een 'issue reference' bevatten met een van zes mogelijke trefwoorden. De aanduiding 'Implementation' van de afgebeelde listing, bijvoorbeeld, betekent dat het gedrag per (compiler) implementatie kan verschillen.

Na de kop van een regel volgt dan een toelichting met een codevoorbeeld. In een commentaar binnen die code staat dan aangegeven wat niet 'compliant' is. Zo'n voorbeeld is er niet altijd; zo vermeldt regel 18-0-1 alleen dat sommige

C++ bibliotheken (zoals <cstdio>) corresponderende C versies hebben (stdio.h). In zo'n geval moet dan de C++ versie worden gebruikt.

Achtergronden

Er worden vijf onzekerheden genoemd rond het gebruik van C++. De ontwikkelaar maakt fouten, de ontwikkelaar begrijpt de taal niet goed, de compiler doet niet wat de ontwikkelaar verwacht, de compiler bevat fouten, en er kunnen run-time fouten optreden.
Het gaat bij MISRA C++ dus niet alleen om onduidelijkheden en beperkingen van de in ISO/IEC 14883:2003 gestandaardiseerde taal, maar om de wisselwerking met de ontwikkelaar. Het doel van de MISRA C++ Working Group was om een deelverzameling van C++ aan te geven die geschikt is voor het gebruik in kritische systemen. MISRA C diende hierbij als voorbeeld.
Het doel was ook om bestaande C++ richtlijnen uit talloze verschillende bronnen in een enkel document samen te brengen, en om nieuwe richtlijnen toe te voegen die de 'state of the art' verder brengen.
Het concrete doel was om één enkele, algemene, verzameling richtlijnen te produceren voor het gebruik van C++ in kritische systemen. Tenslotte moesten die richtlijnen begrijpelijk zijn voor de meerderheid van programmeurs.

MISRA C++ is duidelijk op de praktijk gericht van de ontwikkelaar en niet op die van de projectmanager. Een organisatie kan dan ook niet MISRA C++ compliant zijn, alleen een softwareproduct. Maar het document bevat wel enkele aanwijzingen over het gebruik van de regels, zoals een 'compliance matrix' en de al genoemde 'deviation requests'. De auteurs noemen een mogelijke termijn van een tot twee jaar voordat het gebruik van MISRA C++ volledig is geïmplementeerd binnen een organisatie.
LDRA Software Technology claimt met tools MISRA C++ te ondersteunen en op de website (http://www.misra-cpp.com/) is een aparte sectie ingeruimd voor ondersteunende tools. (Ten tijde van schrijven was dit nog 'Coming Soon'). Er is ook een discussieforum en er is een mailing list.
Het MISRA C++ document is niet in de handel verkrijgbaar. Het kan via de MISRA web site in Engeland worden besteld, als gedrukt boekwerk of (onder licentie) in pdf formaat. ■

Rule 12-8-1 (Required) **A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member**

[Implementation 12-8(15)]

Rationale
If a compiler implementatation detects that a call to a copy constructor is redundant then it is permitted to omit that call, even if the copy constructor has a side effect other than to construct a copy of the object. This is called copy elision.

It is therefore important to ensure that a copy constructor does not modify the program state as the number of such modifications may be indeterminate.

Example

```
class A
{
public:
   A ( A const & rhs )
   : m_i (rhs.m_i )
   {
     ++m_static;    // Non-compliant
   }
private:
          int_32_t m_i;
   static int_32_t m_static;
};
int_32_t A::m_static = 0;
A f ()
{
  return A ();
}
void  b ()
{
  A a = f ( );
}
```

The value that m_static has after the call to b( ) is *implementation-defined.*

Een voorbeeld van een MISRA C++ regel met toelichting en voorbeeld. De nummering aan het begin verwijst naar de betreffende sectie in de ISO/IEC 14882 standaard voor C++ en het laatste cijfer is het volgnummer in het MISRA document. In het voorbeeld wordt, conform regel 3-9-2 (Advisory), *int_32_t* gebruikt in plaats van *int* om duidelijk te maken dat het hier een 32-bits int betreft.