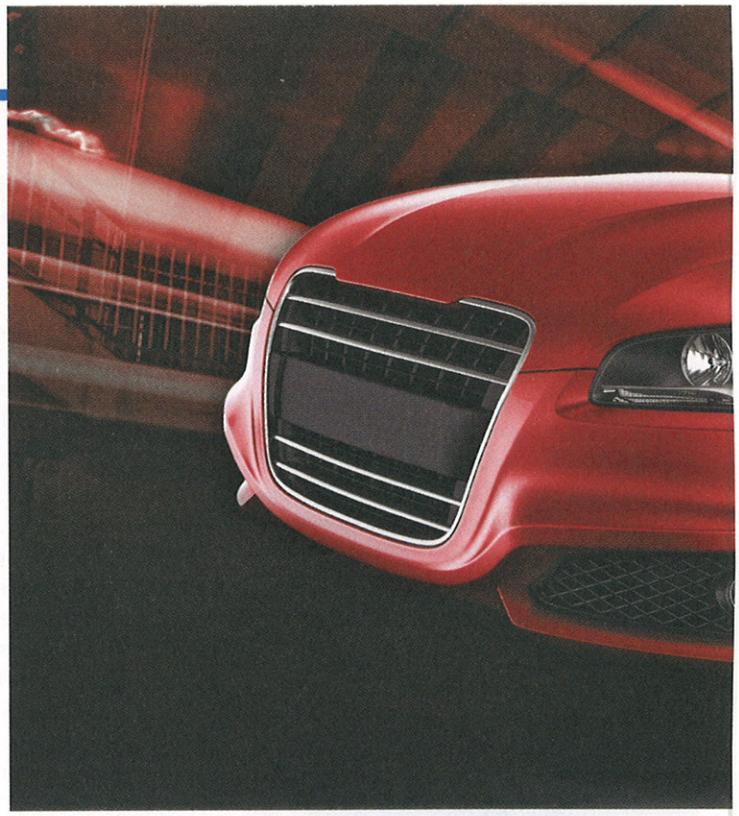


KODIERUNGSSTANDARDS

Warum ein neues MISRA C?

Seit seiner Einführung im Jahr 1998 hat sich MISRA C als Kodierungsstandard etabliert. Ursprünglich gedacht für die Automobilindustrie, haben ihn mittlerweile viele andere Branchen übernommen, wo die Entwicklung von robuster Software zwingend notwendig ist – entweder aus Sicherheitsgründen oder wirtschaftlichen Gründen. Am 18. März 2013 wurde eine neue Version des Kodierungsstandards MISRA C verabschiedet. Warum wurde das nötig? Und was sind die wichtigsten Änderungen?

PAUL BURDEN

Etwas zu ändern, was allgemein akzeptiert und auf breiter Basis eingesetzt wird, ist nicht einfach. Um eine neue Version von MISRA C herauszubringen musste es schon gute Gründe geben. Tatsächlich gab es sogar etliche Gründe: Unterstützung von »C99«, Reaktion auf Rückmeldungen von Anwendern und die Erkenntnis, dass es noch Raum für Verbesserungen gab.

MISRA C:2004 forderte, dass Code dem Standard »C90« entspricht (ISO/IEC 9899:1990, Programming Languages – C), obwohl diese Sprachversion bereits offiziell veraltet war, da sie schon fünf Jahre vorher von der Version C99 abgelöst wurde. C99 erweiterte die Sprache um mehrere nützliche, neue Features, beispielsweise um die »inline«-Funktionen und den Datentyp »_Bool«. Leider kamen dabei auch einige neue Risiken hinzu. Im Jahr 2004 gab es zudem nur wenige Compiler und Werkzeuge, die diese neuer C99-Features unterstützten. Die Entscheidung der MISRA-C-Arbeitsgruppe, C90 weiterhin zu fordern, erschien daher trotz ihrer konservativen Grundtendenz vernünftig. Doch heute, acht Jahre später, hat die Arbeitsgruppe »MISRA C:2012« veröffentlicht, der nun C99 unterstützt.

Gegenüber der Vorversion sind in MISRA C:2012 einige wenige neue Regeln dazugekommen, hauptsächlich C99 betreffend (Schutz vor einigen dieser neuen Risiken), einige wenige wurden entfernt oder umformuliert. C-Code, der zu MISRA C:2004 konform ist, wird wahrscheinlich nach geringfügigen Modifikationen auch die Anforderungen von MISRA C:2012 erfüllen.

Die umfassende Nutzung von MISRA C:2004 über etliche Jahre erbrachte zahlreiche Rückmeldungen von Anwendern und führte zu einer Reihe von Ergänzungen und Verbesserungen. Die Regeln sind jetzt klarer definiert und erläutert, und für ihren Einsatz gibt es bessere Begründungen. Folglich dürfte es weniger Spielraum geben für abweichende Interpretationen durch unterschiedliche Werkzeuge. Jede Regel besteht jetzt aus mehreren Abschnitten:

- Erweiterte Erläuterungen (Amplification) – umfassendere Erklärungen der Regelanforderungen;
- Begründung (Rationale) – Erläuterung, warum die Regel benötigt wird;
- Ausnahmen (Exceptions) – spezielle Situationen, für welche die Anforderungen einer Regel nicht gelten;
- Beispiele (Examples) – Codebeispiele für die Einhaltung der Regeln beziehungsweise den Verstoß gegen die Regeln.

In früheren MISRA-C-Versionen gab es zwei Kategorien von Regeln: erforderliche Regeln und empfohlene Regeln. MISRA-C-konformer Code musste alle erforderlichen Regeln einhalten, sofern nicht eine formelle Abweichung ausgewiesen wurde (eine Abweichung ist eine dokumentierte Begründung für die Nicht-Einhaltung einer Regel). Empfohlene Regeln gelten als flexibler und benötigen keine Feststellung einer formellen Abweichung. Mit MISRA C:2012 gibt es jetzt eine zusätzliche, dritte Kategorie: die zwingend erforderliche Regel (»Mandatory«; Bild 1). Eine solche ist nicht verhandelbar, es sind keine Abweichungen zulässig, da keine denkbaren Situationen eine Nicht-Einhaltung rechtfertigen können.

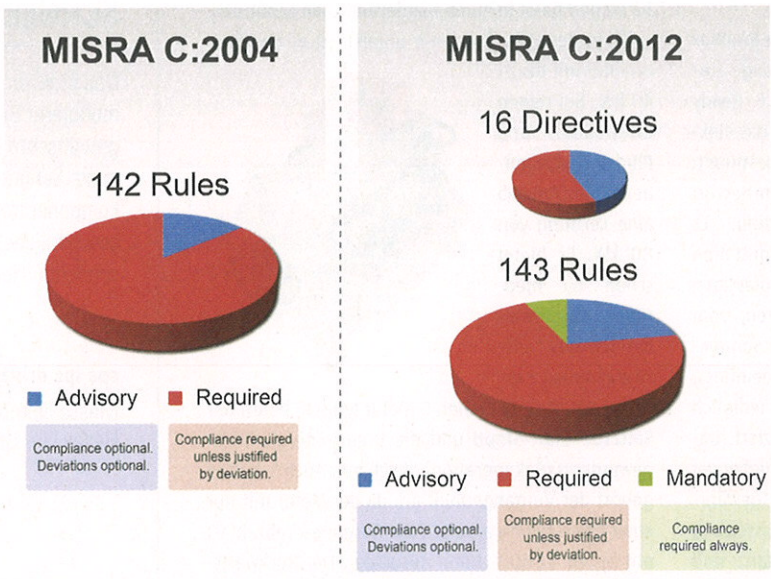
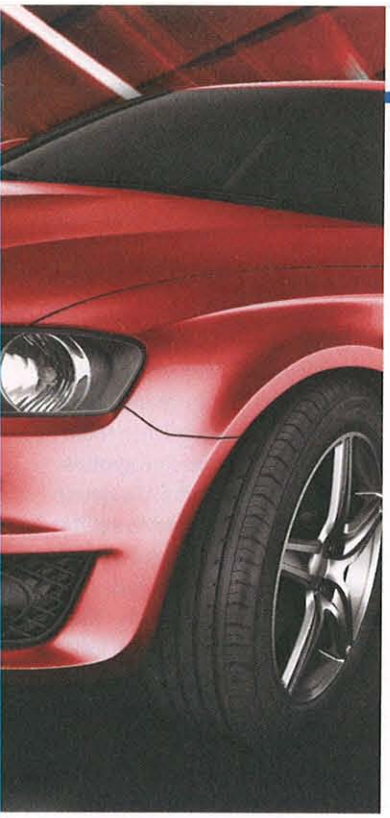


Bild 1: Vergleich von MISRA C:2004 zu MISRA C:2012; neu hinzugekommen sind die Kategorie der zwingend erforderlichen Regeln (Mandatory) und die Anordnungen (Directives), die im Unterschied zu den Regeln möglicherweise nicht präzise definiert sind



Durchsetzbarkeit einer Regel

Ein charakteristisches Merkmal von größter Bedeutung für jede Kodierungsregel ist das Ausmaß, in welchem sich die Regel mithilfe einer statischen Analyse prüfen (durchsetzen) lässt. Die automatische Prüfung (Durchsetzung) von Regeln ist aus diversen Gründen ein bedeutsames Thema:

- Sie spart viel Zeit.
- Sie wirkt sofort, ist zuverlässig, wiederholbar und konsistent.
- Sie verringert die Notwendigkeit, sich auf manuelle Codeanalyse verlassen zu müssen und eliminiert mögliche Peinlichkeiten und Diskussionen

Die MISRA-C-Richtlinien haben immer die Wichtigkeit der statischen Analyse betont, und eines der wesentlichen Merkmale von MISRA C war die Tauglichkeit vieler Regeln für eine automatische Prüfung (Durchsetzung). Dem Problem der Durchsetzbarkeit beziehungsweise automatischen Prüfbarkeit wurde in MISRA C:2012 auf mehreren Wegen Beachtung geschenkt.

Erstens: Man unterscheidet nun zwischen Regeln und Anordnungen; MISRA C:2012 enthält 143 Regeln und 16 Anordnungen (Directives; Bild 1). Regeln sind präzise definierte Anforderungen, die sich ausschließlich durch eine Analyse des Quellcodes durchsetzen lassen. Anordnungen sind möglicherweise nicht präzise definiert, und der Nachweis ihrer Einhaltung kann einen Verweis auf Designdokumentation, Funktionsanforderungen oder ein bestimmtes Maß an subjektiver Beurteilung oder Interpretation erfordern. Beispiele für Anordnungen sind:

- »Codeabschnitte dürfen nicht herauskommentiert werden«.
- »Assembleranweisungen sollen gekapselt und isoliert sein«.
- »Vorsichtsmaßnahmen sollen getroffen werden, damit der Inhalt einer Header-Datei nicht mehr als einmal erscheint«.

Zweitens: Regeln werden eingeteilt als »einzelne Übersetzungseinheit« (Single Translation Unit) oder als »System«-Regeln. Dabei wird der Umfang der Analyse reflektiert, was für den Nachweis der Konformität erforderlich ist. Die Konformität zu manchen Regeln lässt sich aus der Analyse einer einzelnen Übersetzungseinheit ermitteln – oft aus einem kleinen Codeabschnitt. Andere Regeln erfordern womöglich eine erheblich umfassendere Analyse der gesamten Codebasis.

Drittens: Es ist nun bekannt, dass die Einhaltung mancher Regeln niemals garantiert werden kann, und zwar unabhängig davon, wie gut die Regel definiert oder wie leistungsfähig die verfügbare Werkzeugunterstützung ist. Diese Eigenschaft einer Kodierungsregel nennt man »Decidability« (Entscheidbarkeit). Eine entscheidbare Regel ist eine, bei der ein Analysewerkzeug in der Theorie immer zweifelsfrei entscheiden kann, ob die Regel verletzt wurde. Bei einer nicht-entscheidbaren Regel (undecidable) gibt es Situationen, in denen auch eine noch so umfangreiche Analyse keine eindeutige Diagnose liefern kann.

Konformität

Weil der Einsatz von Kodierungsstandards, insbesondere MISRA C heute weit verbreitet ist, ist das Problem der Definition von »Konformität« besonders wichtig geworden. Kodierungsregeln müssen, wo immer möglich, unbedingt

- klar und eindeutig definiert,
- statisch prüfbar und
- entscheidbar sein.

Dies sind einige Grundsätze, die bestimmend für die Entwicklung von MISRA C waren. Dabei handelt es sich um Prinzipien, welche die Entwicklung einer weit verbreiteten *Werkzeugunterstützung für die Durchsetzung der Regeln gefördert haben. Besondere Sorgfalt sollte man auf die Auswahl der Werkzeuge haben. Der MISRA-C-Arbeitsgruppe ist bewusst, dass die Fähigkeiten zur Durchführung einer zuverlässigen und genauen Regeldurchsetzung bei statischen Analysewerkzeugen angesichts einer fehlenden unabhängigen Werkzeugzertifizierung stark variieren. Entsprechend muss der Anwender eine behauptete Konformität zu MISRA C sorgfältig überprüfen. (rh)

PAUL BURDEN

ist Co-Autor des neuen MISRA-C-Kodierungsstandards und leitendes Mitglied der technischen Beratergruppe von PRQA

PRQA
Telefon: 00 44/19 32/88 80 80
www.programmingresearch.com

Wofür Kodierungsstandards?

Organisationen erstellen häufig »Style Guides« mit dem Ziel, die Lesbarkeit und Wartbarkeit von Code durch die Vorgabe eines Standard-Layouts zu verbessern. Kodierungsstandards gehen darüber hinaus. In der Programmiersprache C kann man Code schreiben, der dem Sprachstandard entspricht und sich sogar korrekt kompilieren lässt, jedoch bei der Ausführung unerwartete Folgen haben kann.

Genau dieses Problem gab den Anstoß für das erste MISRA-C-Projekt: Die Erstellung von Richtlinien zur Verringerung des Risikos beim Schreiben von »korrektem«, aber unvorhersehbar reagierendem Code sowie zur Definition einer sicheren Untermenge (Subset) der Programmiersprache C.