MISRA C/C++の最新情報と実践的使用法

MISRA C++ 委員会チェアマン来日講演

MISRA C/C++ 委員会からの最新情報を、レガシーコードやサードパーティにより提供されるソースコードに対する実用的な取り組み方法、ルールからの逸脱の対処とスタンダードへの準拠の証明、セキュリティ対策との関係など、実運用面の話題を絡めて紹介します。また先進の車載ソフトウェア開発ではPOSIX OSやAUTOSAR Adaptive Platformが採用され、C から C++ 言語への移行も急務となっていることを踏まえて、AUTOSAR C++ との統合を含めた MISRA C++ 委員会の活動と MISRA C++ の改訂、公開の計画や流用されたコードの扱いに関して検討されているアイデアも紹介します。

本日の予定

- 13:30～13:40　LDRA社について
- 13:40～14:00　MISRA C:2012 - Deviation and Compliance
  十山 圭介
- 14:00～15:00　MISRA Update 1/2　Chris Tapp
  MISRA C++ の活動状況、
  AUTOSAR C++との関係（統合作業）、
  今後の計画について
- 15:00～15:10　休憩
- 15:10～16:00　MISRA Update 2/2　Chris Tapp
  MISRA Cの状況、
  MISRA Complianceについて、
  新しい考え（コンプライアンスと流用コード）
- 16:00～16:45　質疑応答




LDRA Software Technology

Delivering Software Quality and Security through
Test, Analysis & Requirements Traceability

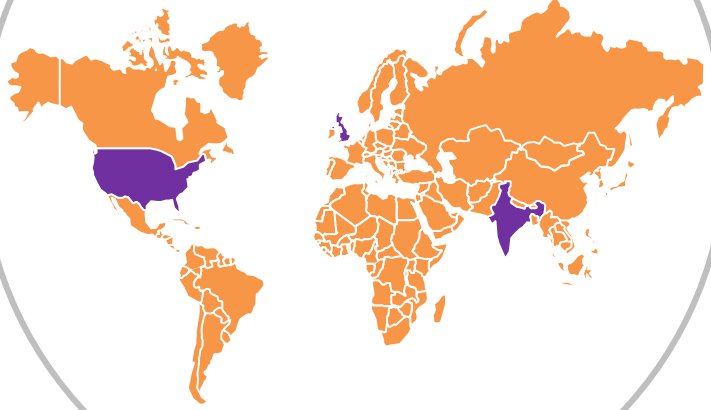
LDRA社テストツール

JSF C++ スタンダードを開発

航空業界のDO-178スタンダード認証に標準的に採用

- コーディングスタンダードチェック
- コード品質の尺度
- カバレッジ解析
- データフロー、コントロールフロー解析
- 単体テスト
  統合テスト
- 要件トレーサビリティ
- テストの管理
- 認証プロセスの支援

LDRA



ISO 9001認証取得

1975年設立 => 40年以上の実績と経験

IEC 61508, IEC 62304, EN 50128,
ISO 26262, IEC 60880 のツール認定取得

スタンダード認証プロセスを支援する
テストツールを提供

各種スタンダード委員会に貢献
e.g. DO-178C, ISO 26262
MISRA C/C++, CERT

SGS TÜV SAAR

CERTIFICATE NO FS/71/220/15/0105 PAGE 1/1

LICENCE HOLDER
LDRA LTD
PORTSIDE, MONKS FERRY,
WIRRAL, CH41 5LH,
UNITED KINGDOM

MANUFACTURING PLANT
LDRA LTD
PORTSIDE, MONKS FERRY,
WIRRAL, CH41 5LH,
UNITED KINGDOM

PROJECT NO-ID: K1C2-AUD1
LICENSED TEST MARK
CERT. REPORT NO.: K1C20003

Tested according to:
IEC 61508:2010 Ed. 2.0
ISO 26262:2011
EN 50128:2011
IEC 60880:2006
IEC 62304:2006

Certified product(s):
LDRA tool suite
LDRAunit
LDRArules
LDRAcover
LDRAlite

Model(s):
Version 9.5

Technical Data and Parameter:
Usable in development of safety-related software acc. to:
- ISO 26262 up to ASIL D, TCL1 can be reached
- IEC 61508 up to SIL 4, class T2 tool
- EN 50128 up to SW-SIL 4, class T2 tool
- IEC 62304 up to SW safety Class C
- IEC 60880

Specific Requirements:
The certificate is based on voluntarily tests. Any changes to the design, components or processing may require repetition of some parts of the qualification in order to retain the certification.
The certificate report is an integral part of this certificate.
The safety-related application guidelines (refer to K1C20003) shall be maintained.

Certification Body for Functional Safety
SGS-TÜV Saar GmbH
Munich, 04.11.2015
Gudrun Neumann

CERTIFICATE

ZERTIFIKAT ◆ CERTIFICATE ◆ СЕРТИФИКАТ ◆ CERTIFICADO ◆ CERTIFICAT

TÜV SÜD Product Service

CERTIFICATE
No. Z10 16 09 84753 003

Holder of Certificate: LDRA Ltd.
Portside, Monks Ferry
Wirral
Merseyside CH41 5LH
UNITED KINGDOM

Factory(ies): 84753

Certification Mark:

Product: Software Tool for Safety Related Development

Model(s): LDRA tool suite
LDRArules
LDRAcover
LDRAunit
LDRAlite

Parameters: The certified tools, classified T2, fulfil the requirements for support tools according to IEC 61508-3 and EN 50128. The tools are qualified to be used in safety-related software development according to IEC 61508, EN 50128 and ISO 26262. It is suitably validated for use in safety-related development according to IEC 62304(ed.1)

The test report is a mandatory part of this certificate.

Tested according to: IEC 61508-3(ed.2)
ISO 26262-8:2011
EN 50128:2011

The product was tested on a voluntary basis and complies with the essential requirements. The certification mark shown above can be affixed on the product. It is not permitted to alter the certification mark in any way. In addition the certification holder must not transfer the certificate to third parties. See also notes overleaf.

Test report no.: LW80043C
Valid until: 2021-10-03
Date, 2016-10-04
Page 1 of 1

TÜV SÜD Product Service GmbH · Zertifizierstelle · Ridlerstraße 65 · 80339 München · Germany

LDRA 社は各種国際スタンダードに貢献

Professor Mike Hennell

Member of SC-205 / WG-71 (DO-178C) formal methods subgroup

Member of MISRA C committee and MISRA C++ committee

Member of the working group drafting a proposed secureC annex for the C language definition (SC 22 / WG14)
ISOとIECの合同委員会

Bill St Clair

Member of SC-205 / WG-71 (DO-178C) Object Oriented Technology subgroup

Andrew Banks

MISRA Committee Chair

Committee Member for Second Edition of ISO 26262

Chair of BSI IST/15/-/26 for Software Testing

MISRA representative to BSI IST/15 for Software and Systems Engineering

Member of BSI IST/15-/20 for Bodies of Knowledge and Professionalization

LDRA 社は各種国際スタンダードに貢献

Dr Clive Pygott

Member of ISO software vulnerabilities working group (SC 22 / WG 23)

Member of MISRA C++ committee

Member of the working group drafting a proposed secureC annex for the C language definition (SC 22 / WG14)

Liz Whiting

Member of MISRA C committee language definition

Chris Tapp

Chairman of MISRA C++ committee

Member of MISRA C committee language definition

スタンダード認証取得を目的に広く採用

コーディングスタンダードへの準拠

- CAST
- CERT
- CMSE
- CONFORM
- CWE
- Customer Sample
- DERA
- EADS
- FSB582-C
- GJB
- HIS
- JPL
- Legacy
- MISRA-AC
- MISRA-C:1998
- MISRA-C:2004
- MISRA-C:2012
- NETRINO
- RUNTIME
- SEC-C
- Standard
- TBrun Requires
- UML
- VSOS
- No Standards Model

- Table A-5 4 - Source Code conforms to standards - Partial - 2 assets, 7 artifacts
  - Code Review Report Artifact Placeholder fulfilled by 1 item
    - Cpp_tunnel_exe.frm.htm
  - Design and coding guidelines document fulfilled by 1 item
    - Misra c 2012.pdf
  - Output: Code Review TCI's (Artifact)
  - Programming Standards Flowgraphs fulfilled by 5 items
    - Flowgraph - Programming Standards for Float_64 TunnelData::Lamp::GetMaximumLumens();
    - Flowgraph - Programming Standards for TunnelData::Lamp::Lamp();
    - Flowgraph - Programming Standards for void TunnelData::Lamp::SendPowerToLamp(const Sint_32 PowerSetting);
    - Flowgraph - Programming Standards for void TunnelData::Lamp::SetLumensOutput(Float_64 LumensRequired);
    - Flowgraph - Programming Standards for Float_64 TunnelData::Lamp::GetMinimumLumens();

LDRA Report File Editor - C:\LDRA_Toolsuite\c\creport.dat*

File Edit View Help

Static | Complexity | Data Flow | Cross Ref | Info Flow | Qual Report | Qualsys | ... | DTMCSFVA

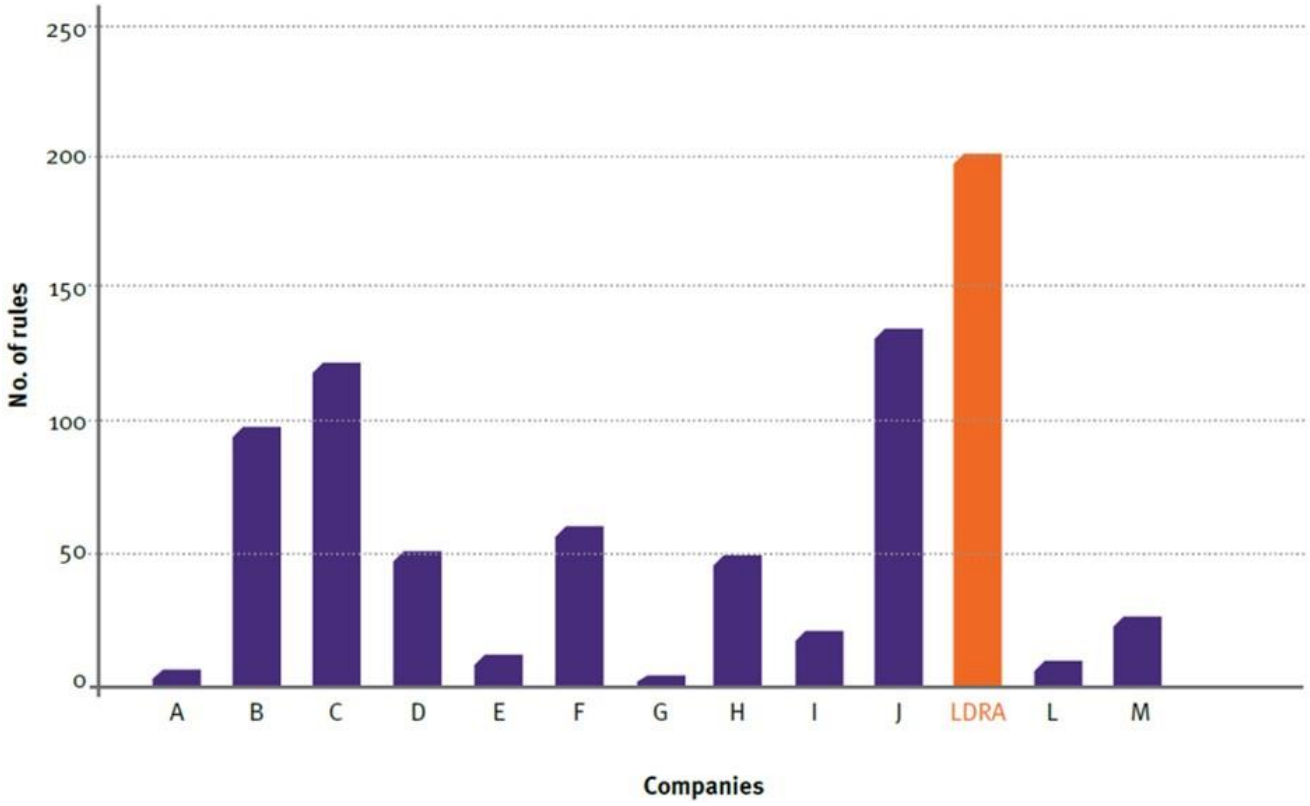
Rule Number	Default Strength	Description	CAS	CERT	WE	HIS	J	L	MISRA-AC	MISRA	MISRA-C 2004	MISRA-C 2012	ETRINO	SEC-C
1	C	Procedure name reused.	☑	☐	☐	☑	☐	☐	☑	☑	☑	☑	☐	☑
2	M	Label name reused.	☑	☐	☐	☐	☐	☐	☑	☑	☑	☐	☐	☐
3	M	More than *** executable reformatted lines in file.	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
4	M	Procedure exceeds *** reformatted lines.	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
5	C	Empty then clause.	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑
6	O	Procedure pointer declared.	☑	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐
7	C	Jump out of procedure.	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
8	C	Empty else clause.	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑

Report File (PC): Section 1 read (24 Models)(24 Modifiers)...Section 2 read (866 Rules)...Section 3 read...Section 4 read...Penalty File (PC): read

Automate coding standards compliance to eliminate defects early in the lifecycle

MISRA-C:1998 ,2004, 2012, MISRA-C++:2008, CERT C, IPA/SEC ESCR

CERT 対応で他を圧倒！



Tool	No. of rules
A	7
B	96
C	122
D	50
E	11
F	63
G	3
H	48
I	21
J	135
LDRA tool suite	**201**
L	10
M	26

複雑度解析やコード品質の尺度

- Table A.12 8 - Limited size and complexity of Functions, Subroutines and Methods - Fulfilled - 1 asset, 1 artifact
  - Code review report fulfilled by 1 item
    - Mingw_c_cashregister.frm.htm
  - Design and coding guidelines document fulfilled by 1 item
    - Complexity_Guidelines from Acme_coding_standard.doc

Code Quality View (Maintainability)

Procedure Calls	Cyclomatic Complexity	Knots
GetOptimum	98 : (Fail)	114 : (Fail)
LzmaDec_DecodeReal	82 : (Fail)	76 : (Fail)
LzmaDec_TryDummy	61 : (Fail)	94 : (Fail)
GetOptimumFast	41 : (Fail)	38 : (Fail)
LzmaDec_DecodeToDic	30 : (Fail)	57 : (Fail)
LzmaEnc_CodeOneBlock	30 : (Fail)	32 : (Fail)
main2	23 : (Fail)	28 : (Fail)
LzmaEncProps_Normalize	20 : (Fail)	7 : (Fail)
Decode2	14 : (Fail)	15 : (Fail)
Hc4_MatchFinder_GetMatches	14 : (Fail)	9 : (Fail)
Bt4_MatchFinder_GetMatches	14 : (Fail)	9 : (Fail)
MatchFinder_Create	14 : (Fail)	8 : (Fail)
GetMatchesSpec1	13 : (Fail)	14 : (Fail)
LzmaEnc_Alloc	13 : (Fail)	8 : (Fail)
BtGetMatches	11 : (Fail)	11 : (Fail)
LzmaEnc_SetProps	11 : (Fail)	2
SkipMatchesSpec	10	13 : (Fail)
MatchFinder_ReadBlock	10	12 : (Fail)
Hc_GetMatchesSpec	10	12 : (Fail)

Zone.cpp

Clarity	100%
Executable ref. Lines	147
Depth of Loop Nesting	1
Total LCSAJs	46
Unique Operands	105
Average Length of Basic Blocks	4.03%
Comments in Headers	53
Maintainability	100%
Unreachable Branches	0
Unreachable Lines	0
Maximum LCSAJ Density	4
Unreachable LCSAJs	0
Total LCSAJs	46
Vocabulary	119
Cyclomatic Complexity	9
Knots	23
Essential Cyclomatic Complexity	1
Testability	100%
Fan Out	7
File Fan in	0
Number of Procedures	7
Procedure Exit Points	1
Number of Loops	5
Unreachable Branches	0
Unreachable Lines	0
Maximum LCSAJ Density	4
Unreachable LCSAJs	0
Total LCSAJs	46
Total Operands	286
Number of Basic Blocks	36
Executable reformatted Lines	145
Cyclomatic Complexity	9

Quickly assess code quality to reduce cost of development and verification

データカップリング、コントロールカップリング

- Table A-7 8 - Test coverage of software structure (data coupling and control coupling) is achieved - Fulfilled - 2 assets
  - Software Verification Results fulfilled by 2 items
    - Callgraph - Pass/Fail Coverage
    - DataCouplingReport.html

Requirement based Test case

Variable I/O View

Value	Name	Type
I CALCULATE_CMD	command	S_U16
I *** Value Retained ***	airspeed	S_U32
O 0	airspeed	S_U32

Unexecuted code for the given test case

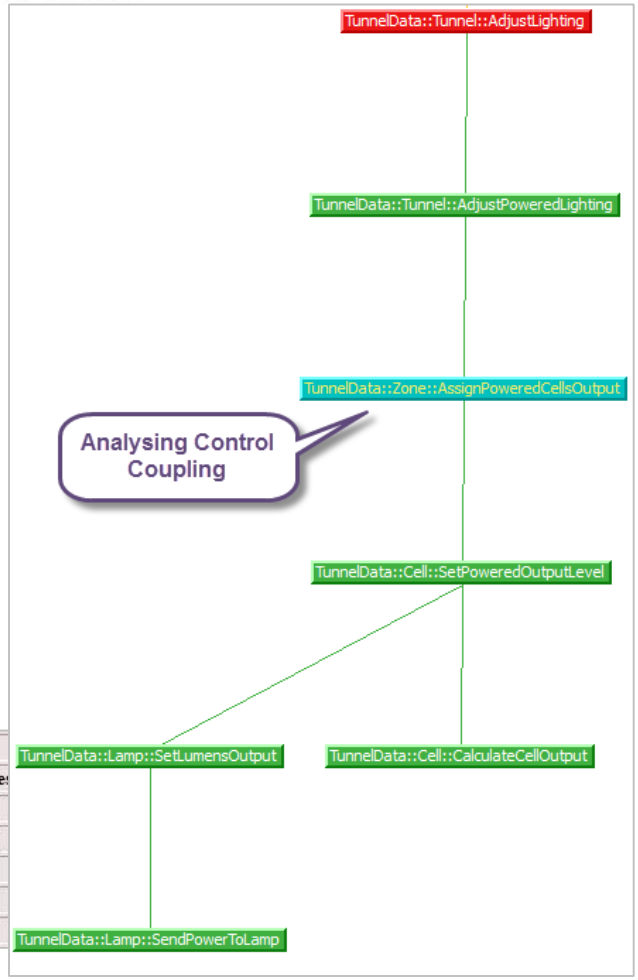
```
31  void runAirspeedCommand  (S_U16 command)
32  {
33      switch(command)
34      {
35          case CALCULATE_CMD:
36          calculateAirspeed (airspeed);
37          break;
38          case DISPLAY_CMD:
39          displayAirspeed (airspeed);
40          break;
41      }
42  }
```

```
void
runAirspeedCommand (
S_U16 command )
{
 switch (
  command
 )
 {
  case 1 :
   calculateAirspeed ( airspeed ) ;
   break ;
  case 2 :
   displayAirspeed ( airspeed ) ;
   break ;
 }
}
```

Unexecuted data reference for the given test case

Variable Name	File	Procedure	Type Code	Attribute Code	Used on lines
airspeed	AirspeedCommands.cpp	runAirspeedCommand	G	R	39 *****
					36
command	AirspeedCommands.cpp	runAirspeedCo			31
					33
factor	AirspeedCalculate.cpp	calculateAirspeed			16

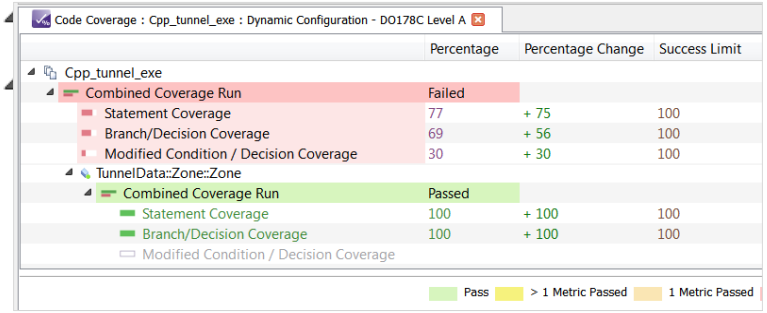
On line 39 the reference to airspeed by displayAirspeed is not executed with this test case

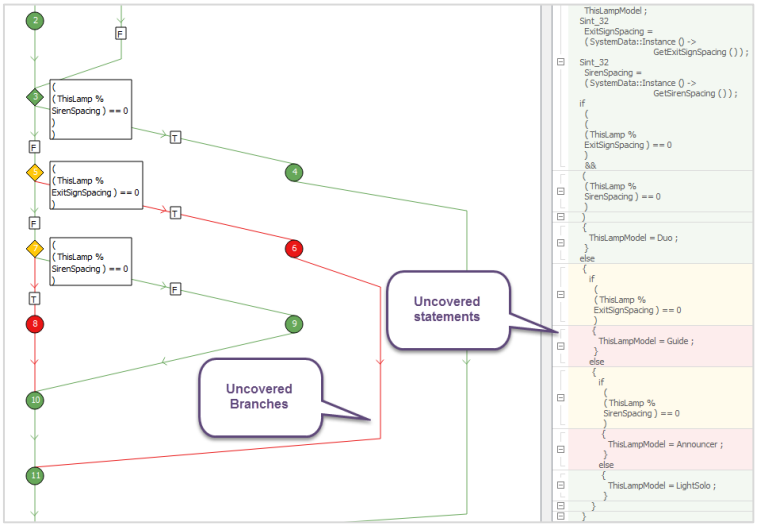


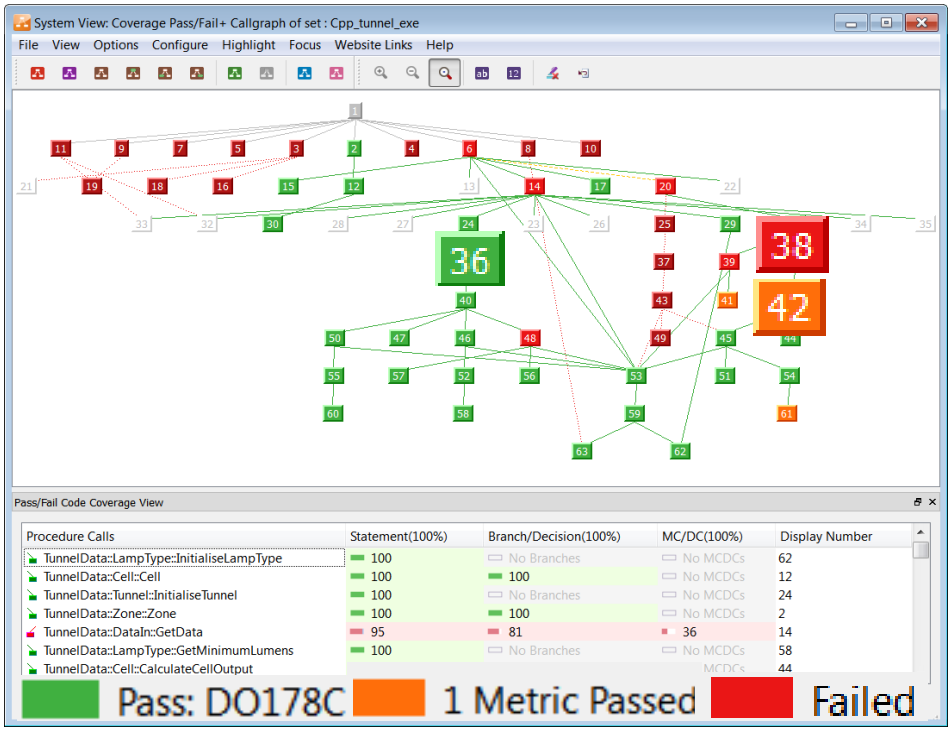
Dramatically reduce the effort required to achieve data and control coupling objectives

構造カバレッジ解析

- Table A-7 7 - Test coverage of software structure (statement coverage) is achieved - Partial - 1 asset, 3 artifacts
  - Code Coverage Callgraphs fulfilled by 1 item
    - Callgraph - Pass/Fail + Coverage for Cpp_tunnel_exe
  - Code Coverage Flowgraphs fulfilled by 1 item
    - Flowgraph - Pass/Fail + Coverage for void TunnelData::Cell::SetPoweredOutputLevel(const Float_64 LumensDemandPerMetre, const Float_64 CellSpacing);



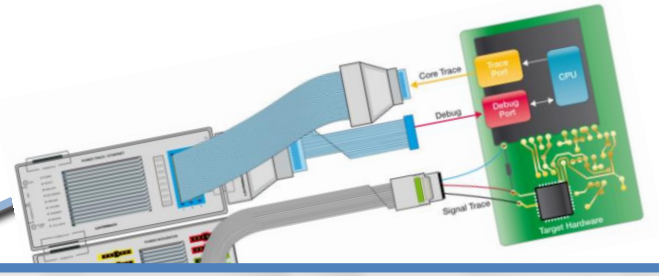


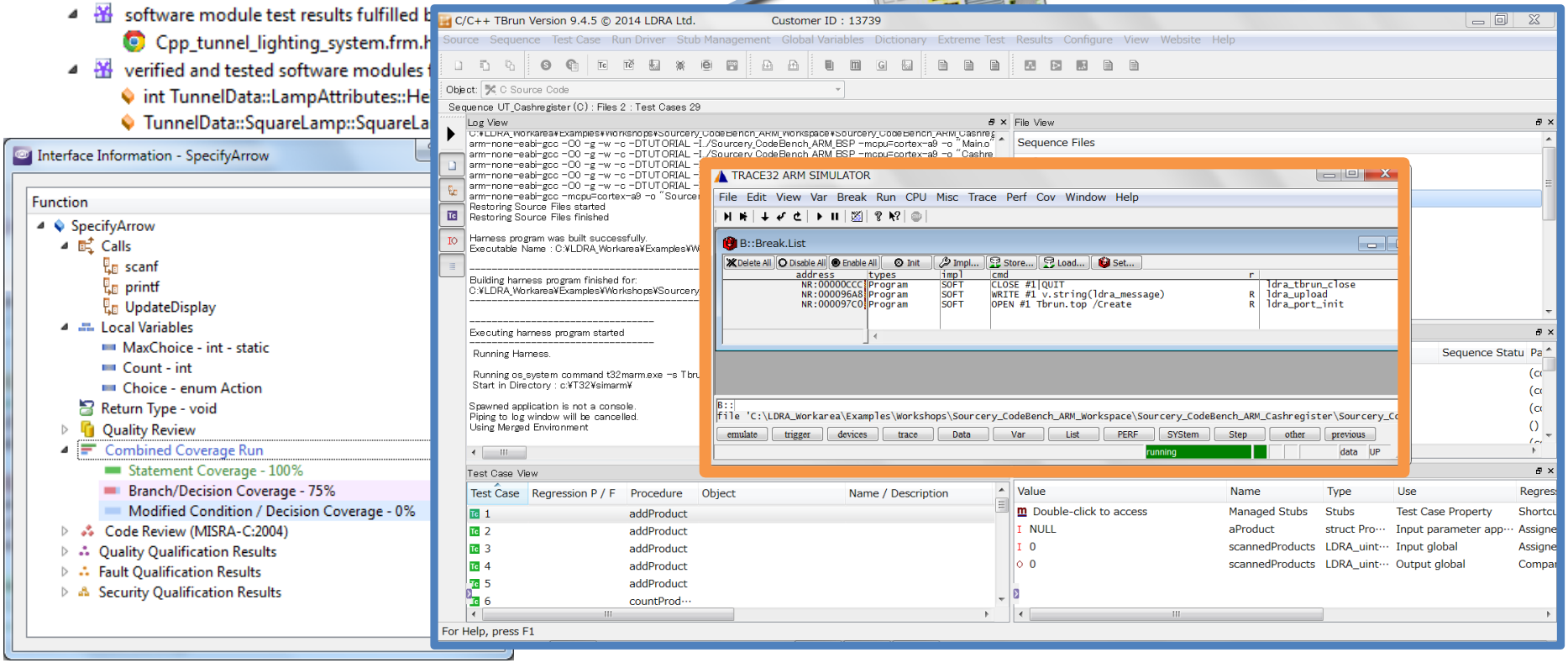


Automate all levels of code coverage across a wide array of embedded environments

単体テスト：あらゆるターゲット環境で実行

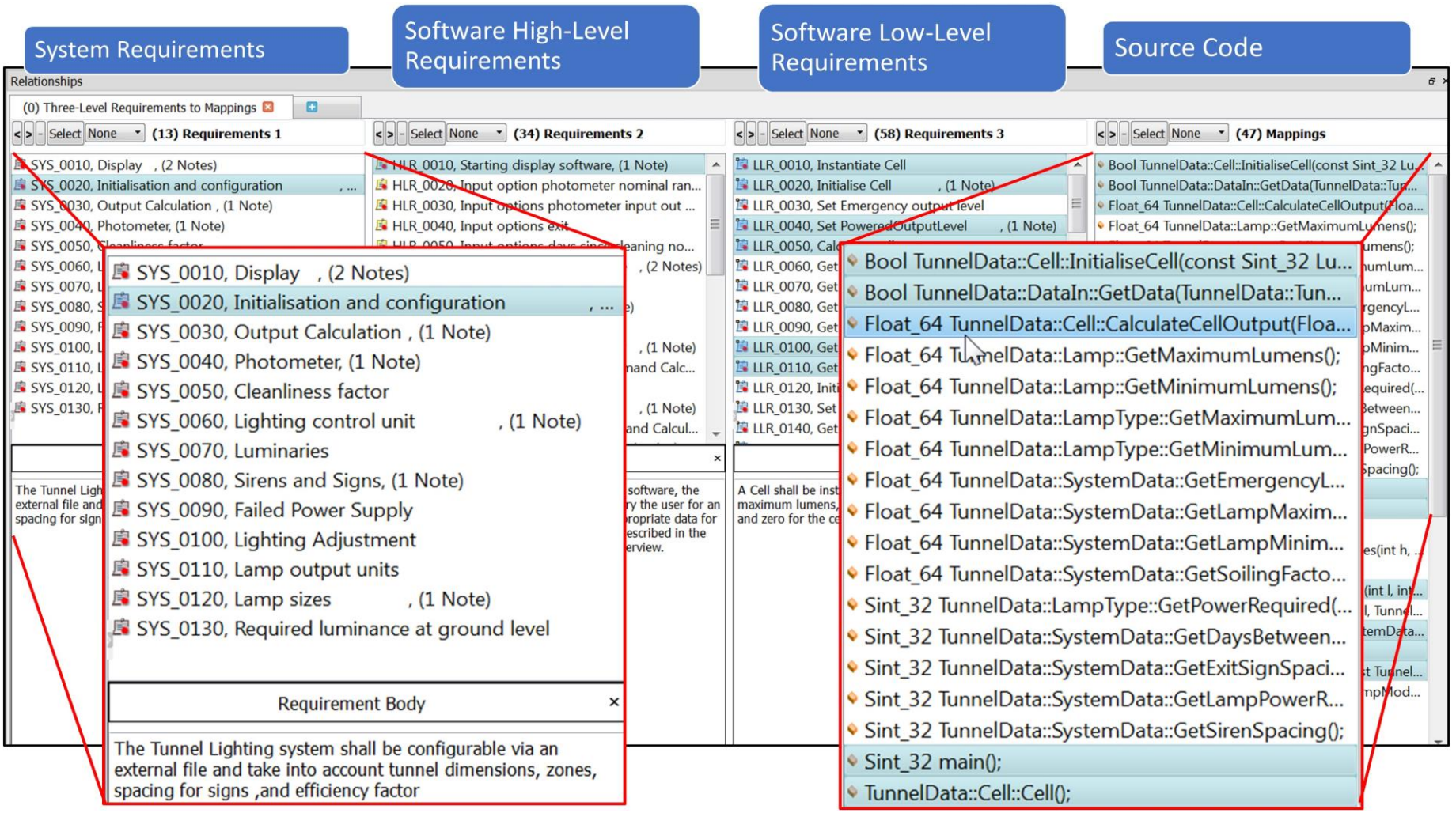
- 7.4.7 - Software design and development - Software module testing
  - Input Assets
    - software module test specification fulfilled by 1 item
      - Low_Level_Tests.xlsx
    - Code review report fulfilled by 1 item
      - Cpp_tunnel_lighting_system.frm.htm
  - Output Artifacts
    - software module test results fulfilled b
      - Cpp_tunnel_lighting_system.frm.h
    - verified and tested software modules f
      - int TunnelData::LampAttributes::He
      - TunnelData::SquareLamp::SquareLa





Achieve structural coverage objectives, verify low level requirements and greatly reduce verification costs

双方向のトレーサビリティ



要件～コード、テストまで変更のインパクトを成果物間で追跡

For Further Information:

FUJI SETSUBI

www.fuji-setsu.co.jp
info@fuji-setsu.co.jp

LDRA

www.ldra.com
info@ldra.com

@ldra_technology

LDRA Software Technology

LDRA Limited

MISRA C:2012 - Deviation and Compliance

Chris Tapp

十山 圭介
富士設備工業株式会社 技術部

2019年11月15日

逸脱とコンプライアンス

- MISRAコンプライアンス
- 違反と逸脱
- 管理

コンプライアンスとは？

- プロジェクトのコードがMISRAサブセット（MISRA C:2012など）で課される制約や規制を遵守していることの表明

- 自己認証の形態をとる
  - コードを作成する組織が準拠していることを保証する責任を負う
  - コンプライアンスを主張するために根拠が必要

開発者サイドでの処理手続き

Appendix F　Process and tools checklist

付録 F にあるチェックリスト

This Appendix provides a checklist of the development process and tool use guidance that need to be followed in order to claim MISRA C compliance as described in Section 5.3.

Section	Guidance
3.1	A choice has been made between C90 and C99
4.2	There is a process for dealing with deficiencies in the C implementation
5.3.1	The translator has been configured to accept the correct version of the C language
5.3.1	The translator has been configured to generate an appropriate level of diagnostic information
5.3.1	The translator has been configured appropriately for the target machine
5.3.1	The translator's optimization level has been configured appropriately
5.3.3	There is a process for investigating and resolving any diagnostic messages produced by the translator
5.3	There is a compliance matrix showing how compliance with each MISRA C guideline is to be checked
5.3.2	The analysis tools have been configured to accept the correct version of the C language
5.3.2	The analysis process can deal with any language extensions that have been used
5.3.2	The analysis tools have been configured for the implementation, for example to be aware of the sizes of the integer types
5.3.3	There is a process for investigating and resolving any diagnostic messages produced by the analysis tools
5.4	There is a deviation process for recording and approving deviations
5.2.1	There is a process for ensuring that the program has sufficient resources, such as processing time and stack space
5.2.1	There is a process for demonstrating and recording the absence of run-time errors, for example in module designs

処理手続きと生成物

5.5 Claiming compliance

Compliance cannot be claimed for an organization, only for a project.

When claiming compliance with MISRA C for a project, a developer is stating that evidence exists to show:

1. A compliance matrix has been completed which shows how compliance has been enforced;
   コンプライアンスマトリクスを完成させる
2. All the C code in the project is compliant with the guidelines of this document, or is subject to approved deviations;
3. There is a record of each approved deviation;
4. The guidance given in this section, as well as that given in Section 3 and Section 4, has been adopted;
5. Staff are suitably skilled and have a sufficient range of experience.

Note: when claiming that code complies with the guidelines, the assumption is being made that the tools or reviewers have identified all non-compliances. Since other tools or reviewers might identify different non-compliances, it is important to recognize that compliance claims are not absolute, but depend on the checking process used.

A summary of the guidance referred to in point (4) is provided in the form of a checklist in Appendix F.

処理手続きと生成物

Guideline	Compilers		Checking tools		Manual review
	'A'	'B'	'A'	'B'	
Dir 1.1					Procedure x
Dir 2.1	no errors	no errors			
…					
Rule 4.1			message 38		
Rule 4.2				warning 97	
Rule 5.1	warning 347				
…					

5.5 Claiming cor

Compliance cannot be cla

When claiming complianc
show:

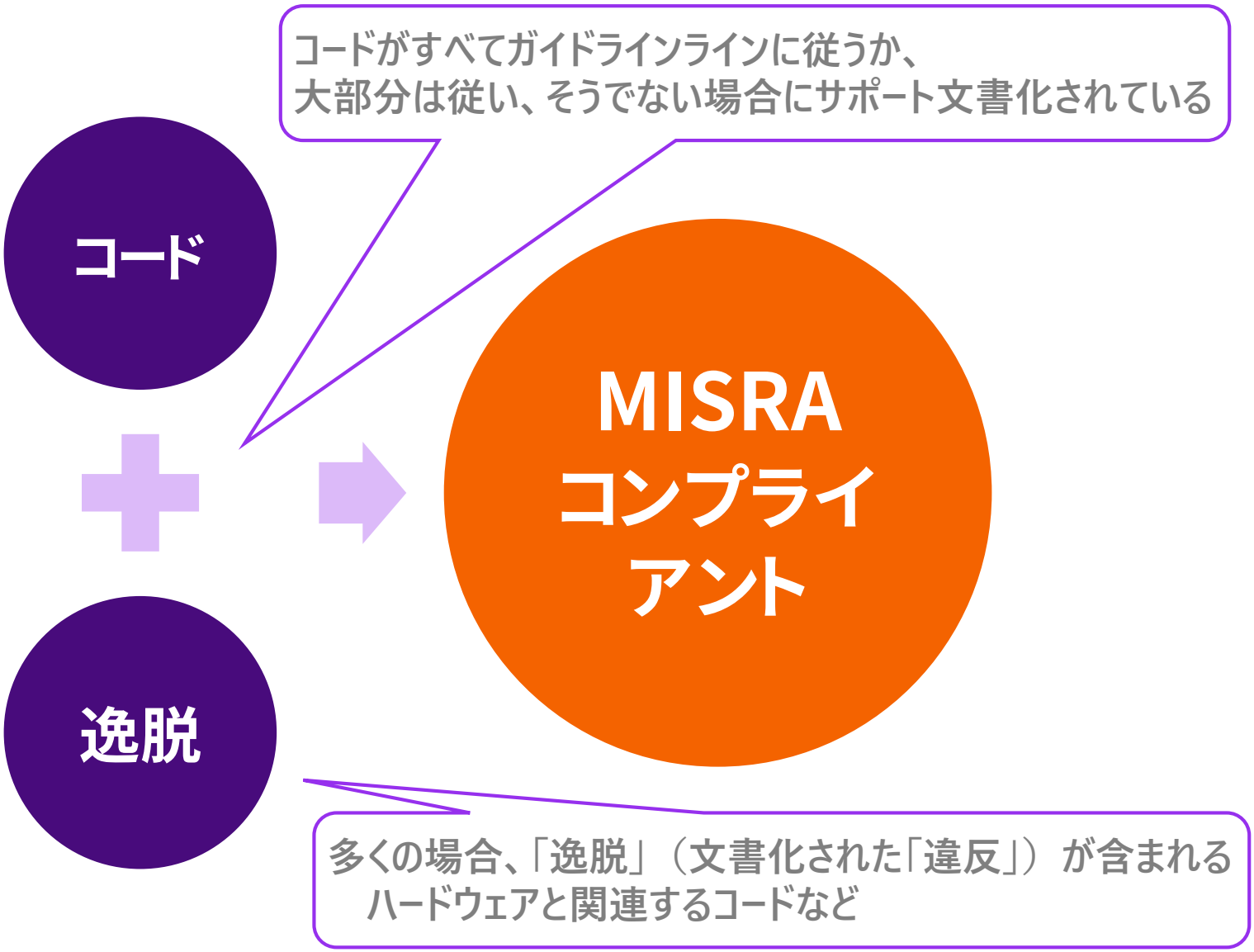
1. A compliance matrix has been completed which shows how compliance has been enforced;
2. All the C code in the project is compliant with the guidelines of this document, or is subject to approved deviations;
3. There is a record of each approved deviation;
4. The guidance given in this section, as well as that given in Section 3 and Section 4, has been adopted;
5. Staff are suitably skilled and have a sufficient range of experience.

コンプライアンスマトリクスを完成させる

Note: when claiming that code complies with the guidelines, the assumption is being made that the tools or reviewers have identified all non-compliances. Since other tools or reviewers might identify different non-compliances, it is important to recognize that compliance claims are not absolute, but depend on the checking process used.

A summary of the guidance referred to in point (4) is provided in the form of a checklist in Appendix F.

簡単にいうと、MISRAコンプライアントとは



しかし、課題がある

「逸脱」の使用では、「違反」を許すことで起こりうるリスクの存在を考慮したものではない場合もありえる

逸脱	• 承認者は？ • 良い理論的根拠があるか、便利さか？ • 監査のコストは？
リスク	• 請負者が導入する • 顧客は容認できるか？
緩和	• 技術的に完全か？ • 開発者に理解されるか？ • 他のモジュールとの係わりは？

どのようにしてコンプライアンスを検証するか？

- 取得者の受入れプロセスと品質保証監査の一部であるべき
- 開発プロセスは監査されるべき
- 検査プロセスは詳細に検討されるべき
- 逸脱はレビューされるべき
- Etc
- しかし、コンプライアンスはほとんど検証されていない！

製品に責任を持つ部門がコンプライアンスを検証する

検証によって開発プロセスが適切なものであると保証するには・・・

主張の多くが妥当でないか、証拠に依っていない

どのようにしてコンプライアンスを測るか？

- 一般に、供給者によるほとんど根拠のない単なる（最小の）表明だけ – コンプライアンスの主張はそのまま信用されるべきもの

  根拠があったとしても、主張をサポートする生成物の監査には尺度が重要

- MISRA Compliance:2016では必要とされるものを結びつけている
  - 標準の文書セットを作成しなければならない
  - コンプライアンスの主張は証拠に基づかなければならない
  - 「逸脱によるコンプライアンス」は将来に多くの努力を要する！

以前のMISRAでは詳細が記載されていなかった
→
MISRA Compliance:2016で明確にした

MISRA Compliance:2016

- 現MISRAガイドライン（1998、2004、2008、2012）を置き換えるオプション、将来のすべてのバージョンでは必要とされる
- コンプライアンスの主張に要求されるもの、およびプロジェクト内でのガイドライン適用方法を明確にする
- コンプライアンス主張を行うのに必要なものを定義する

MISRA Guideline Enforcement
MISRAガイドラインの施行

- MISRAガイドラインでは、それぞれ最小限の施行区分を設定
  - Mandatory, Required, Advisory
    （必須）　（必要）　（推奨）

- この区分はガイドライン違反であるかどうか、「逸脱」が必要であるかを定義する

区分	施行処置
必須	「違反」は決して許されない
必要	「違反」は「逸脱」があれば許される
推奨	「違反」は識別されるが「逸脱」は不要

コードの欠陥を招くリスクが大きいので、より多くの文書、プロセスが必要

MISRA Compliance:2016 – ガイドライン施行計画書（GEP）

プロジェクトで作成

- ガイドラインへのコンプライアンスのチェック法を示す
- 以前、コンプライアンスマトリックスとよばれていたもの

追加のサポート情報が必要

- ツールのバージョン
- データまたは設定ファイル
- ツール起動でのオプション
- ツールで違反検知が可能である証拠
- 手動でのチェック作業があれば、その詳細

コンパイラ、解析ツールの出力メッセージ、手動チェックの結果

MISRA Compliance:2016 – GEP Example

MISRA C:2012やMISRA C++: 2008のコンプライアンスマトリックスと同じ

ガイドライン	コンパイラ	解析ツール	手動チェック
Dir 1.1			Procedure “A”
Dir 2.1	エラー報告なし		
		...	
Rule 4.1		Message 38	
Rule 4.2		Message 97	
Rule 5.1	Warning 347		
		...	
Rule 12.1		Message 79	
Rule 12.2		Message 452	Procedure “B”
Rule 12.3		Message 103	
Rule 12.4		Message 27	

MISRAコンプライアンスのplanは同じ構造をもつ

MISRA Compliance:2016 – MISRAガイドライン再分類

ガイドライン達成は、レビューしてプロジェクトに適応させるものだが、考慮されていないことが多い

できるだけ多くのガイドラインを「必須」にする

- 一般に、ほとんどの「必要」ガイドラインはプロジェクト内で「必須」とするべき
- 違反が正当化されることがあり得るので、それらは「必須」とはされていない

多くのガイドラインを「必要」レベルにとどめ、「逸脱」できるようにした

「推奨」ガイドラインのレベルを変更する

- プロジェクトに有益な強い区分へ上げる
- 「推奨」ガイドライン(// コメントの使用 など)違反すべて発見する必要がない
  - このことを許すため、「非適用」区分を導入

変更点： コンプライアンスのチェックを行わない

MISRA Compliance:2016 – ガイドライン再分類計画書（GRP）

プロジェクトでどのようにガイドラインを適用するかレビュー

デフォルトのガイドライン区分のレビュー

- プロジェクト開始時に取得者のコンプライアンス要求を考慮して着手

似た既存プロジェクトを利用することも可

  - 違反が禁止される場合、「必要」ガイドラインを「必須」と再分類
  - 違反が逸脱を要求する場合、「推奨」ガイドラインを「必要」と再分類

得られたGRP

- 取得者が逸脱を受け入れられる箇所を示す
- 開発者が逸脱を考慮する箇所を示す

MISRA Compliance:2016 – ガイドラインの再分類

- 再分類後に許可される区分はMISRAガイドラインで設定される分類によって変わる

MISRA区分	変更後の区分			
必須	必須			
必要	必須	必要		
推奨	必須	必要	推奨	非適用

- 再分類によりMISRAガイドラインで設定するレベルは、元が「推奨」の場合に限り、低くできる

MISRA Compliance:2016 – GRP Example

ガイドライン	MISRAの区分	変更後の区分
Dir 1.1	必要	必須
Dir 2.1	必要	必要
	•••	
Rule 4.1	必要	必要
Rule 4.2	推奨	非適用
Rule 5.1	必要	必須
	•••	
Rule 12.1	推奨	必須
Rule 12.2	必要	必要
Rule 12.3	推奨	推奨
Rule 12.4	推奨	必要

強い制約を課している

MISRA Compliance:2016 – 「逸脱」と「逸脱許可」

「逸脱」の役割

- MISRAコンプライアンスの考えは供給者と取得者間でコンプライアンスをどのように解釈するかの理解が必要
- 「逸脱」により、不可避な「違反」が明確に定義されたプロセスによって認可され、逸脱報告書によってサポートされる

MISRA Compliance:2016 –「逸脱」と「逸脱許可」

「逸脱」の考えを拡張して、その作業の大部分を複数のプロジェクトや組織で共有しやすくする

逸脱許可

- 「逸脱許可」は基本的に、逸脱がプロジェクトや組織内で再利用されたり、その情報をサードパーティで共有したりするために必要なバックグラウンド作業ができるようにするテンプレート
- 供給者が、プロジェクト内で必要になるかもしれないと想定する逸脱のタイプを宣言して、取得者が「原則的に」承認できるようにするもの
- 逸脱報告書とは異なるが、逸脱報告書から参照されるもの

MISRA Compliance:2016 – ユースケースと「逸脱許可」

特定のユースケースに対する「逸脱」の要求を文書化する

ユースケース

- 関連するガイドラインが「逸脱」を必要とするかもしれないという特定の条件を識別する

逸脱許可

- ユースケースを形式化する
- 強力な根拠に支えられている（パフォーマンスなど）
- セーフティ/セキュリティの維持を保証するのに必要な特定の緩和策の詳細を与える

逸脱許可の例

Rule 13.7 Boolean operations whose results are invariant shall not be permitted.

関連ルールの見出し

識別子：作成者、ルール、許可のバージョン

Permit / MISRA / C:2004 / 13.7.A.1
The result of a logical operation is invariant in a particular build configuration.

逸脱サポートのためのユースケース

Reason Code quality (Reusability)

Background

許可される理由の説明、詳細

Invariant operations are often introduced when software is designed to be built under different configurations for a product line. For example, a logical operation which is "always true" in a particular configuration may evaluate to "false" in a different build.

When developing source code which is to be configured in different ways, it may be preferable to tolerate some logical operations which are invariant in a particular build configuration in order to make the code easier to understand and maintain.

Requirements

1. The reason for the invariant operation shall be justified and documented.

Notes

1. The presence of the invariance will introduce *unreachable code* in violation of Rule 14.1, requiring a deviation supported by Permit / MISRA / C:2004 / 14.1.A.1. It is acceptable to use a single deviation record to cover the Rule 13.7 and Rule 14.1 violations.

MISRA Compliance:2016 – 「逸脱許可」の使用 – まとめ

- 供給者が、プロジェクト内で「逸脱」が必要となりそうなユースケースを特定し、それらが原則として取得者にとって受け入れ可能であるか見つけることは良い考えである
  - コードが届けられたときの「サプライズ」を防ぐ助けとなる
  - 「逸脱許可」は逸脱を行う意図の事前通告手段を提供し、導入前に原則として使用受け入れを得るために使用される

MISRA Compliance:2016 – 「逸脱」の正当化

逸脱は、違反をコード内で許可することが正当である場合にのみ受け入れられる

逸脱が受け入れられない場合は？

- 単に開発者にとって便利になる
- 他に合理的な選択肢が可能な場合
- 使用した場合のより広い影響を考慮していない
- サポートプロセスがない
- 所定の技術的権限者の承諾がない

『ポインタで指し示された型からconst修飾やvolatile修飾を取り除くキャストを行ってはならない(Rule 11.8)』に対する逸脱があったとすると、『型が「const修飾された文字型へのポインタ」型のオブジェクト以外に文字列リテラルを代入してはならない』(Rule 7.4) による保護を消してしまう

MISRA Compliance:2016 – 「逸脱」の正当化

逸脱が合理的である場合は？

– 以下の4つの「理由」が定義されている

- ISO/IEC 25010で定義されるコード品質が向上する場合：
  - 時間効率性、フォールトトレランス、モジュラリティ、再利用性 など
- ハードウェアへのアクセスが必要な場合
- MISRAコンプライアントな流用コードを統合した場合
- 非コンプライアントな流用コードが使用された場合

MISRA Compliance:2016 – ネイティブコードと流用コード

ネイティブコード

- プロジェクトのスコープ内で開発されたソース
- 必要に応じてコンプライアントにできる

必要な変更はプロジェクトで定義された品質管理プロセスに則っている

流用コード

- プロジェクトのスコープ外で開発されたもの
- ソースまたはバイナリ形式で提供される
- 一般にインターフェースとしてヘッダーファイルを使用する
- 通常、コンプライアントにはできない
  - 非コンプライアントが必ずしも「悪い」を意味しない

サードパーティのライブラリやレガシーコードを含む

コードの変更ができない

MISRA Compliance:2016 – 流用コードによる違反

ネイティブコード内での違反は管理しやすいが、流用コードが係ると簡単ではない

プロジェクト内では非コンプライアントかもしれない

- MISRAコンプライアントとして書かれてはいない
- MISRAの違うバージョンではコンプライアント

非コンプライアントの原因

- ヘッダーファイルの内容と使用（オブジェクト、宣言、マクロ展開 など）
- 別々ではコンプライアントなモジュール間の干渉（識別子名の衝突など）

問題

- 「必須」や「必要」と再分類されたガイドラインがプロジェクト内で違反される

MISRA Compliance:2016 – 流用コードによる違反

「必須」ガイドライン違反の対策 – 2つある

「必須」ガイドラインの違反

- ネイティブコードを修正して違反を除去できるかもしれない
- GRPがレビューでき、「逸脱」を許すように制限を緩めることができる
  - 関連するガイドラインが妥当なMISRAガイドラインにおいて「必須」ではない!
  - MISRAガイドラインで必須なガイドラインの場合、流用コードの使用は不可能!

「必須」に分類されるガイドラインの違反があれば、プロジェクトは実行不可能
→ 流用コードの変更を考えることが必要だが、重大な欠陥に関係するかもしれず開発者の助けが必要になる

MISRA Compliance:2016 – 流用コードによる違反

「必要」ガイドライン違反の対策 – 2つある

「必要」ガイドラインの違反

- ネイティブコードを修正して違反を除去できるかもしれない
- 「逸脱」の使用が受け入れられるかもしれない
  - 新しい「逸脱許可」の導入が必要 ー 流用コードに2つの「理由」が存在
  - 「逸脱」の使用があれば、取得者と合意したコンプライアンス戦略と両立する

MISRA Compliance:2016 – 流用コード内の違反

ネイティブコードと同様に管理するが、違いがある、

「必要」ガイドラインの違反

- 「逸脱」プロセスによって表明しなければならない
- 正当化は、単にコードが変更不可であるかもしれない（そして正しいと分かっている）
- セーフティ/セキュリティが維持されていることを証明しなければならない

各流用コードについて

- 違反のユースケースをカバーする逸脱許可が必要となるかもしれない
- その流用コード使用に関する場合のみ、追加の逸脱が受け入れ可能

MISRA Compliance:2016 – ガイドラインコンプライアンスサマリー(GCS)

4つ目の生成物
ガイドラインへの対応のまとめ

GCSでプロジェクトのコンプライアンスレベルを記録する

- 主張は流用コードを含むプロジェクト全体に対して
- 標準フォーマットによって、色々なプロジェクトのコンプライアンス主張間でより意味のある比較が可能
- 各ガイドラインに対してコンプライアンスレベルが宣言される

主張するレベル	意　　味
コンプライアント	プロジェクトに「違反」はない
逸脱	「逸脱」によってサポートされた「違反」
違反	サポートされない「違反」
非適用	コンプライアンスに対するチェックはなされない

許可されるコンプライアンスレベルは変わる

- MISRAガイドラインで設定される区分に依存する

「コンプライアント」のみ

区分	許可されるコンプライアンスレベル			
必須	コンプライアント			
必要	コンプライアント	逸脱		
推奨	コンプライアント	逸脱	違反	非適用

「推奨」だけが「非適用」に降格できる

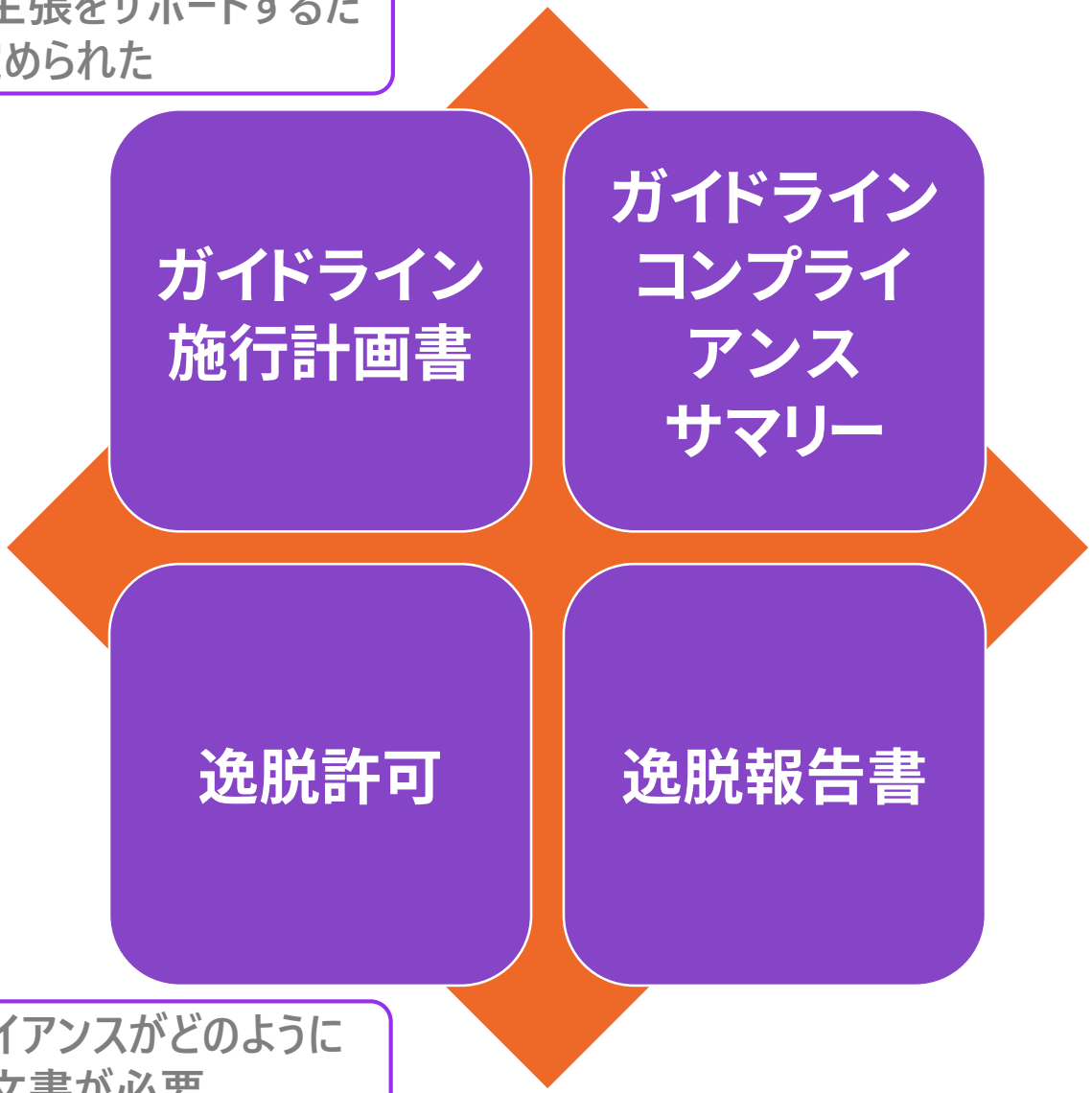
- 宣言されるレベルは全プロジェクトで最悪のコンプライアンスを反映する – 例：ネイティブコードは「コンプライアント」で流用コードが「逸脱」の場合、全体として「逸脱」になる

MISRA Compliance:2016 – GCS Example

ガイドライン	MISRA区分	コンプライアンス
Dir 1.1	必要	コンプライアント
Dir 2.1	必要	逸脱
	•••	
Rule 4.1	必要	逸脱
Rule 4.2	推奨	非適用
Rule 5.1	必要	コンプライアント
	•••	
Rule 12.1	推奨	コンプライアント
Rule 12.2	必要	逸脱
Rule 12.3	推奨	違反
Rule 12.4	推奨	逸脱

MISRA Compliance:2016 – プロジェクト引き渡し

コンプライアンス主張をサポートするための生成物が定められた



加えて、コンプライアンスがどのように達成されたかの文書が必要

Conclusions

- 「逸脱」は強力なツールだが、注意深く使用しなければならない
- MISRA C:2012の「逸脱」プロセスは正しく使用するとうまく働く
- MISRAコンプライアンスは「逸脱」プロセスを形式化する
  - 供給者の義務と取得者の期待に対する共通の理解
  - 正当化されない「逸脱」の数と付随するコストを減らす
  - MISRAのどのサブセットにも適用できる
  - 標準のプロセスがより広く使用される

MISRA Update

Presented by
Chris Tapp
Chairman MISRA C++
(Chris.Tapp@LDRA.com)

Tokyo, 15th November 2019

Contents

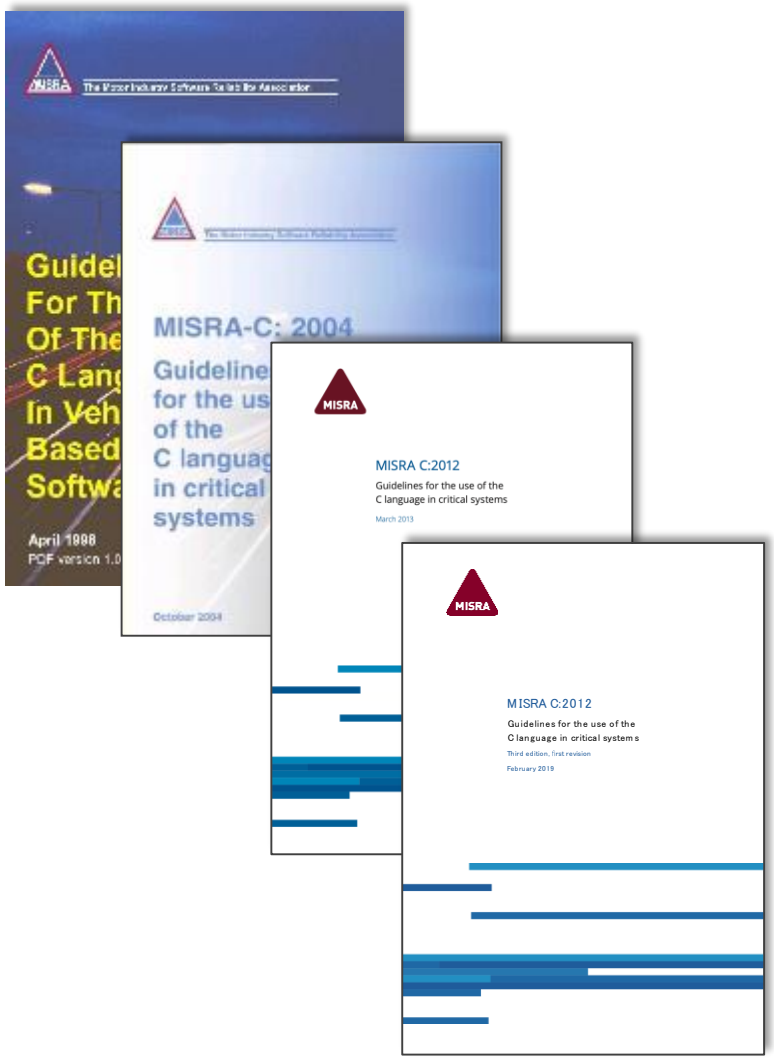
- MISRA C++ – Activity, AUTOSAR, Roadmap
  MISRA C++ ー 活動、AUTOSAR、ロードマップ
- MISRA C – Activity, Roadmap
  MISRA C ー 活動、ロードマップ
- MISRA Compliance – Activity
  MISRAコンプライアンス ー 活動
- Future ideas – Compliance and Adopted Code
  将来への見解 ー コンプライアンスと流用コード

MISRA

A Quick History

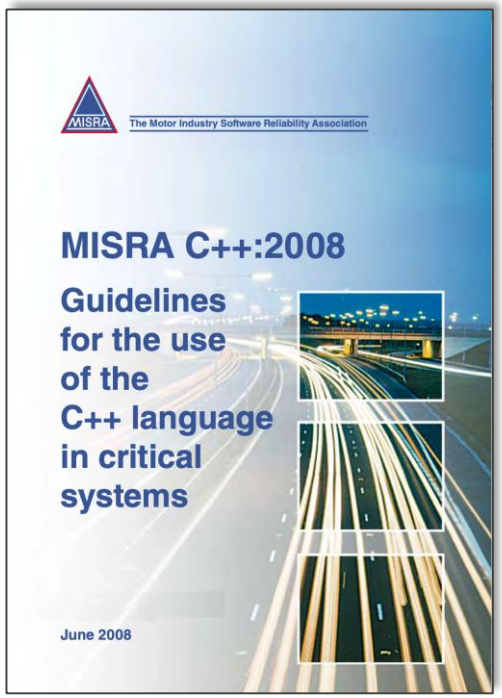
A Quick History – MISRA C

- MISRA-C:1998 (aka *MISRA-C1*)
  - "Guidelines for the use of the C language in vehicle based software"
  - Compatible with ISO/IEC 9899:1990 (aka C90)
- MISRA-C:2004 (aka *MISRA-C2*)
  - "Guidelines for the use of the C language in critical systems"
  - Remains compatible with ISO/IEC 9899:1990 (aka C90)
- MISRA C:2012 (aka *MISRA-C3*)
  - Adds compatibility with ISO/IEC 9899:1999 (aka C99)



MISRA C++ – A Quick Summary

- MISRA-C++:2008
  - “Guidelines for the use of the C++ language in critical systems”
  - Compatible with ISO/IEC 14882:2003



A Quick History – The Vision

- The vision of MISRA C and C++:
  - The MISRA Guidelines define subsets of a standardized languages in which the opportunity to make mistakes is either removed or reduced.
    MISRAガイドラインは標準規格言語のサブセットを定義するもので、間違いを犯す機会をなくしたり減らしたりするもの
  - Many standards for the development of *safety-related software* require, or recommend, the use of a language subset, and this can also be used to develop *any application with high integrity or high reliability requirements*.
    安全関連ソフトウェア開発のための多くの標準では、言語サブセットの使用が必要か、または推奨されており、それを使用して*高い整合性や高い信頼性要件を持つどのようなアプリケーション*を開発することも可能
- So, they are not just for automotive or safety-critical projects
  ということで、言語サブセットの使用は自動車やセーフティクリティカルなプロジェクトだけのものではない

A Quick History – MISRA Compliance

- A standalone document, available as a free download, defining what is required to claim “MISRA compliance”
  「MISRAコンプライアンス」を主張するために必要なものを定義した、ダウンロードして無料で利用可能なスタンドアロンドキュメント
- Compatible with all existing and future versions of MISRA C and MISRA C++
  MISRA C、およびMISRA C++の既存、将来のすべてのバージョンと互換
- Optional for:
  - MISRA C up to and including Ed 3, Rev 1
  - MISRA C++:2008
- Mandatory for all future versions of MISRA C and MISRA C++
  MISRA C、およびMISRA C++の将来のすべてのバージョンで必須

MISRA C++

Activity

MISRA C++ – Past Activity

- June 2008 — Publication of MISRA C++:2008 …
  … Followed by a long period of inactivity due to lack of members
  発行後、メンバー不足により長期間非活動

- July 2014 — Group reformed with the aim of … 次の目的でグループ再編
  … Improving communication コミュニケーションの改善
  … Adding support for C++14 C++14のサポートを追加
  … General improvements in document quality (c.f. MISRA C)
  ドキュメント品質の全般的な改善（MISRA Cを参照）
  … Adding guidance for use of dynamic memory and others
  動的メモリの使用などに関するガイダンスの追加
  … Considering the use of function contracts / annotations
  関数コントラクト/アノテーションの使用を検討

MISRA C++ – Past Activity

- April 2017 — Contact established with AUTOSAR … AUTOSARとコンタクト確立
  - … Face-to-face meeting in June 2017 2017年6月に対面会議
  - … Groups agreed to co-ordinate future efforts
    将来の取組みを調整することに合意
  - … AUTOSAR C++ would be merged into next MISRA C++
    おそらくAUTOSAR C++は次のMISRA C++に統合される
  - … Work started on integration 統合作業の開始
  - … Work suspended in some other areas to make this the priority
    これを優先するため、他のいくつかの領域で作業を中断
- October 2018 — AUTOSAR C++ 18-10 release available … AUTOSAR C++ 10-18 リリース
  - … Formal handover from AUTOSAR to MISRA
    AUTOSARからMISRAへ正式な引渡し
  - … Further integration work to review changes over 17-03 release
    17-3 リリース以降の変更をレビューするためのさらなる統合作業
- October 2019 — AUTOSAR Review Completed … AUTOSARレビューの完了
  - … Modifications of MISRA C++:2008 rules analysed
    MISRA C ++:2008ルールに対する変更の分析
  - … New rules added by AUTOSAR reviewed
    AUTOSARが追加した新ルールのレビュー
  - … MISRA C:2012 improvements of benefit to C++ identified
    C++に有益なMISRA C:2012での改善の特定

MISRA C++

AUTOSAR

MISRA and AUTOSAR

- Joint Press Release 29th January 2019
  - AUTOSAR has released their C++ guidelines as part of the Adaptive AUTOSAR platform twice a year since March 2017. This paved the way for the E/E development with the focus on performance as well as safety and security.
    AUTOSARは、2017年3月から年2回の頻度でAdaptive AUTOSARプラットフォームの一部としてC++ガイドラインをリリース。これにより、パフォーマンス、安全性とセキュリティに重点を置いたE/E開発への道が開かれた
  - MISRA will merge the AUTOSAR guidelines with its own established best practice to develop a single “go to” language subset for safety-related C++ development. The MISRA led guidelines will incorporate the latest version of C++ language - C++17 - and, when available, its successor C++20.
    MISRAは、AUTOSARガイドラインを自身の確立したベストプラクティスに統合し、安全関連のC++開発用の単一の頼りになる言語サブセットを開発する。MISRA主導のガイドラインには、C++言語の最新バージョン（C++17）と、利用可能な場合はその後継であるC++20が組み込まれる
- There will be no further development of AUTOSAR C++, but team members are still active as key players have joined the MISRA Working Group.
  AUTOSAR C++のさらなる開発はないが、主要プレーヤーはMISRAワーキンググループに参加しているため、チームメンバーは活動を継続

AUTOSAR

Guidelines for the use of the C++14 language in critical and safety-related systems
AUTOSAR AP Release 18-10

Document Title	Guidelines for the use of the C++14 language in critical and safety-related systems
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	839

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	18-10

Document Change History			
Date	Release	Changed by	Description
2018-10-31	18-10	AUTOSAR Release Management	• Added traceability for ISO 26262 (B.6) • New rules resulting from continued analysis of the C++ Core Guideline • Finished addressing MISRA review comments of the 2017-03 release • Improvements of already existing rules, more details in the Changelog (D.3) • Marked the specification as obsolete
2018-03-29	18-03	AUTOSAR Release Management	• New rules resulting from the analysis of JSF, HIC, CERT, C++ Core Guideline • Improvements of already existing rules, more details in the Changelog (D.2) • Covered smart pointers usage • Reworked checked/unchecked exception definitions and rules
2017-10-27	17-10	AUTOSAR Release Management	• Updated traceability for HIC, CERT, C++ Core Guideline • Partially included MISRA review of the 2017-03 release • Changes and fixes for existing rules, more details in the Changelog (D.1)

1 of 510

Document ID 839: AUTOSAR_RS_CPP14Guidelines
— AUTOSAR CONFIDENTIAL —

AUTOSAR – Document review conclusions
AUTOSARドキュメントのレビュー結果

- Some rules are the same as those in MISRA C++:2008
  いくつかのルールは、MISRA C++:2008のルールと同じ
- Some rules are similar to those in MISRA C++:2008
  MISRA C++:2008のルールと似ているルールもある
- Some rules have major differences within AUTOSAR
  いくつかのルールはAUTOSARと大きな違いがある
- New rules have been added by AUTOSAR
  AUTOSARによって新しいルールが追加
  - Most are related to language sub-setting　大部分は言語のサブセットに関連
  - Some are related to software design　ソフトウェア設計に関連するもの
  - Some are related to code style　コードスタイルに関連するもの
  - Some are related to the software development process
    ソフトウェア開発プロセスに関連するもの

AUTOSAR – Rules with major differences

- Differences had to be identified, analyzed and understood
  違いを特定、分析、理解する必要
  - Some changes have been rejected
    一部の変更は拒否され
  - Many have been accepted
    多くが受け入れられた
  - Some have lead to ideas for new MISRA guidelines – which may not appear in the next update …
    新しいMISRAガイドラインのアイデアにつながるものもあったが、次の改訂では現れないかもしれない

AUTOSAR – New Rules

- Those related to software design, code style and software development process will generally not be included within the next version of MISRA C++
  ソフトウェア設計、コードスタイル、ソフトウェア開発プロセスに関連するものは、通常、MISRA C++の次のバージョンには含まれない
  - They are not needed to define a language sub-set, so it would be unreasonable for MISRA to apply their restrictions globally
    言語サブセットの定義には必要でないため、MISRAがその制限をグローバルに適用するのは道理に合わない
  - AUTOSAR has the option to maintain and enforce these rules
    AUTOSARには、これらのルールを維持し施行するオプションがある
- The concerns covered by the other new rules will be supported by the next version of MISRA C++
  他の新しいルールがカバーする関心事は、MISRA C++の次のバージョンでサポートする
  - AUTOSAR will be able to mandate the use of this version once it has been published
    AUTOSARは、このバージョンが公開されると、その使用を義務付けることができる

Integration Tasks

- Update existing MISRA C++ rules
  既存のMISRA C++ルールを更新
  - Structure similar to that in MISRA C:2012
    MISRA C: 2012と同様の構造
  - Review any changes made by AUTOSAR
    AUTOSARによる変更をレビュー
- Add new rules from AUTOSAR
  AUTOSARから新しいルールを追加
  - Ensure they are consistent with MISRA terminology
    MISRAの用語と矛盾しないことを確認
- Editorial
  - Editorial consistency, …　編集の一貫性
  - Not technical work, but it takes a lot of time!
    技術的な作業ではないが、時間がかかる！

Provisional Timescales
暫定的タイムスケール

- January 2020 — Feature complete 主要機能完了
- April 2020 — Draft for public comment
  パブリックコメントドラフト
- July 2020 — Review and act on feedback
  フィードバックのレビューと対応
- October 2020 — Final draft ready for publication
  最終ドラフトの公開準備完

MISRA C++

Roadmap

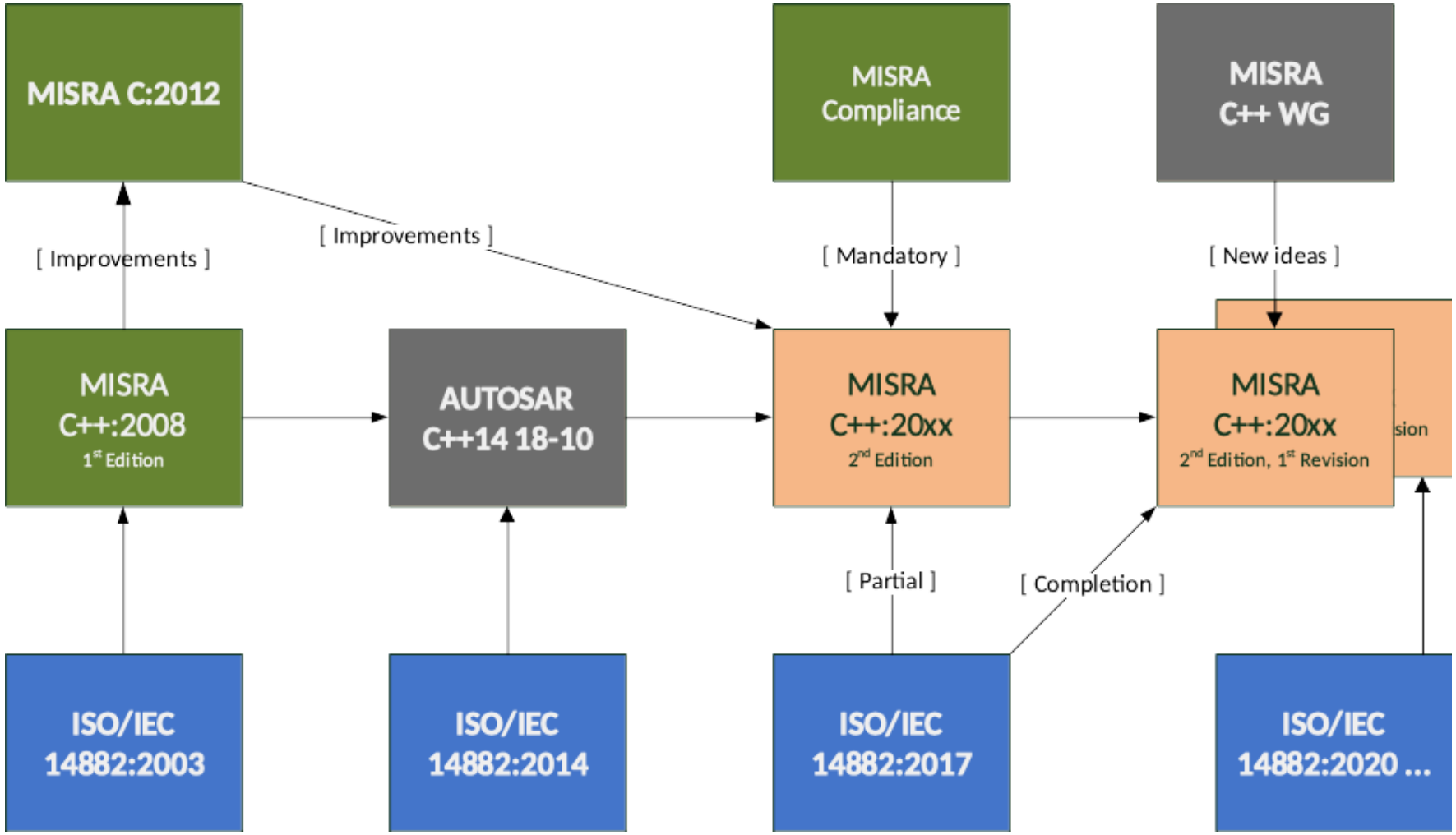
MISRA C++ – Future Plans

- A series of incremental updates to MISRA C++:20XX is proposed:
  MISRA C++:20XXの一連の増分更新を提案
  - Improvement in treatment of Standard and Template Libraries
    標準およびテンプレートライブラリの処理の改善
  - More consideration given to hosted applications
    ホストされたアプリケーションに対する追加の考慮事項
  - Support for future updates to the C++ language
    C++言語の将来の更新のサポート
  - Increased coverage of undefined and unspecified behaviours
    未定義および未規定動作のカバレッジの増加

- Supporting documents: サポートドキュメント
  - Mappings to CERT C++, ISO 24772
    CERT C++、ISO 24772との対応

- Development of “Best Practice” guidance documents:
  「ベストプラクティス」なガイダンス文書の作成
  - Use of dynamic memory 動的メモリの使用
  - Class design (value, resource management, base classes, …)
    クラス設計（値、リソース管理、基底クラスなど）
  - Exception handling 例外処理
  - + others, where proscriptive “rules” are not the ideal solution
    その他、規制的な「ルール」が理想的な解でない場合

MISRA C++ – Future Plans



MISRA C

Thanks to

Andrew Banks

Chairman MISRA C

(Andrew.Banks@LDRA.com)

for assistance with this section

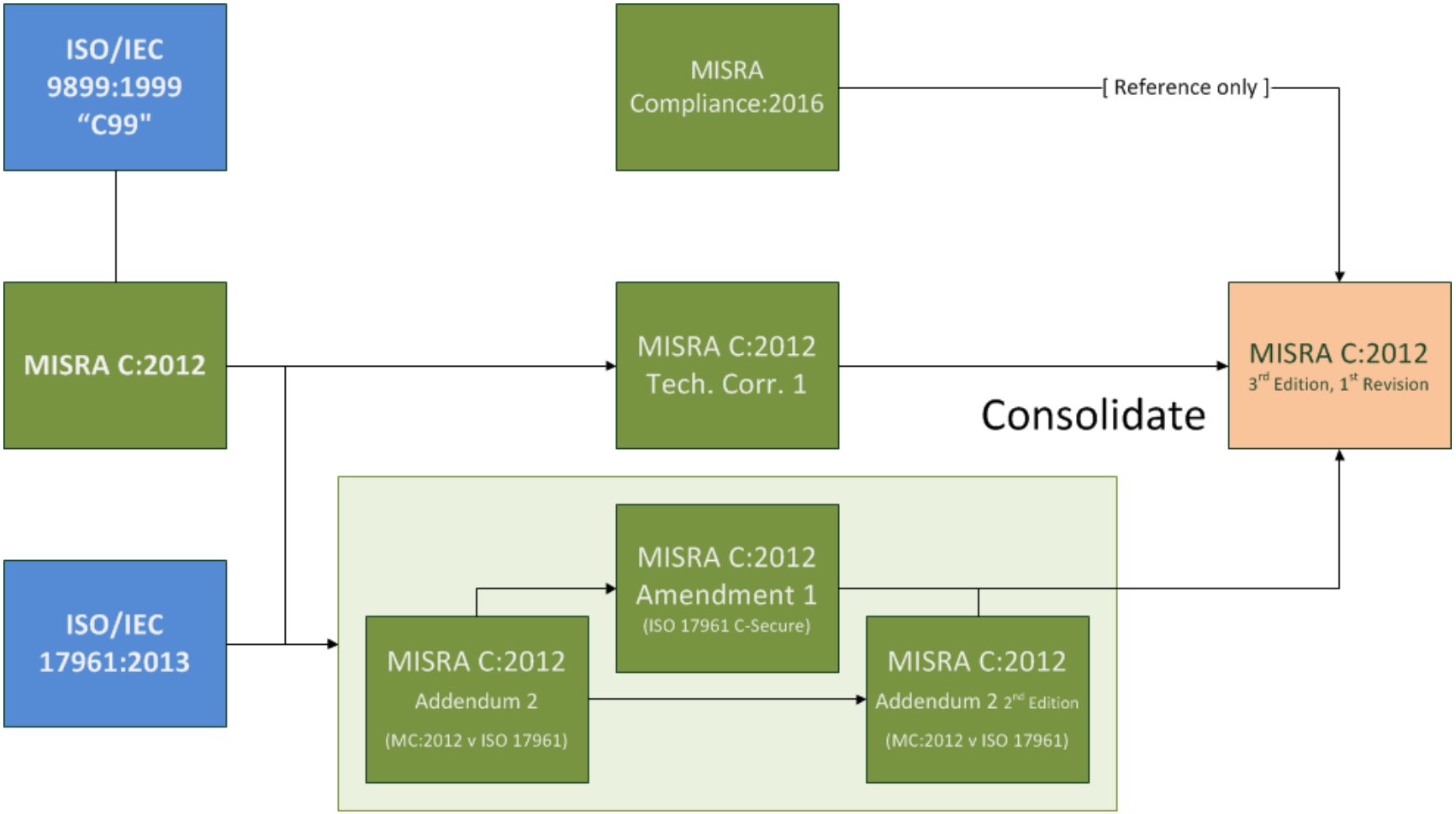
MISRA C

Activity

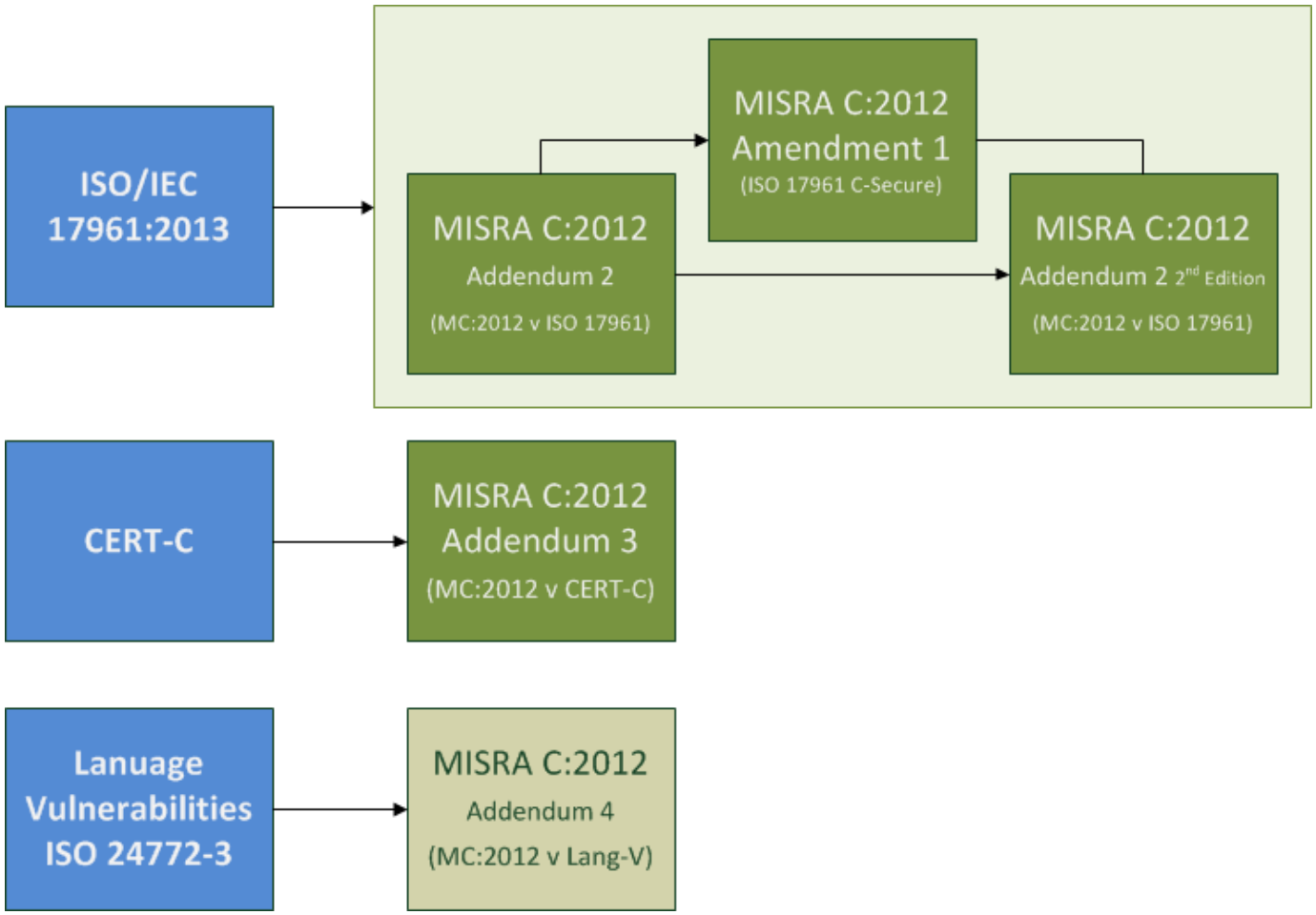
MISRA C – Past Activity

- March 2013 Publication of MISRA C:2012
- April 2016 Publication of ...
  ... MISRA Compliance
  ... MISRA C:2004 Permits
  ... MISRA C:2012 Addendum 2 (MISRA C v ISO 17961:2013)
  ... MISRA C:2012 Amendment 1 (Additional Security Guidelines)
- June 2017 Publication of MISRA C:2012 TC1
- January 2018 Publication of ...
  ... MISRA C:2012 Addendum 2 (2nd Edition)
  ... MISRA C:2012 Addendum 3 (MISRA C:2012 v CERT C)
- February 2019 Publication of MISRA C:2012 (3rd Edition, 1st Revision)

MISRA C – Current State



Supporting Studies



MISRA C

Roadmap

C11 and C18

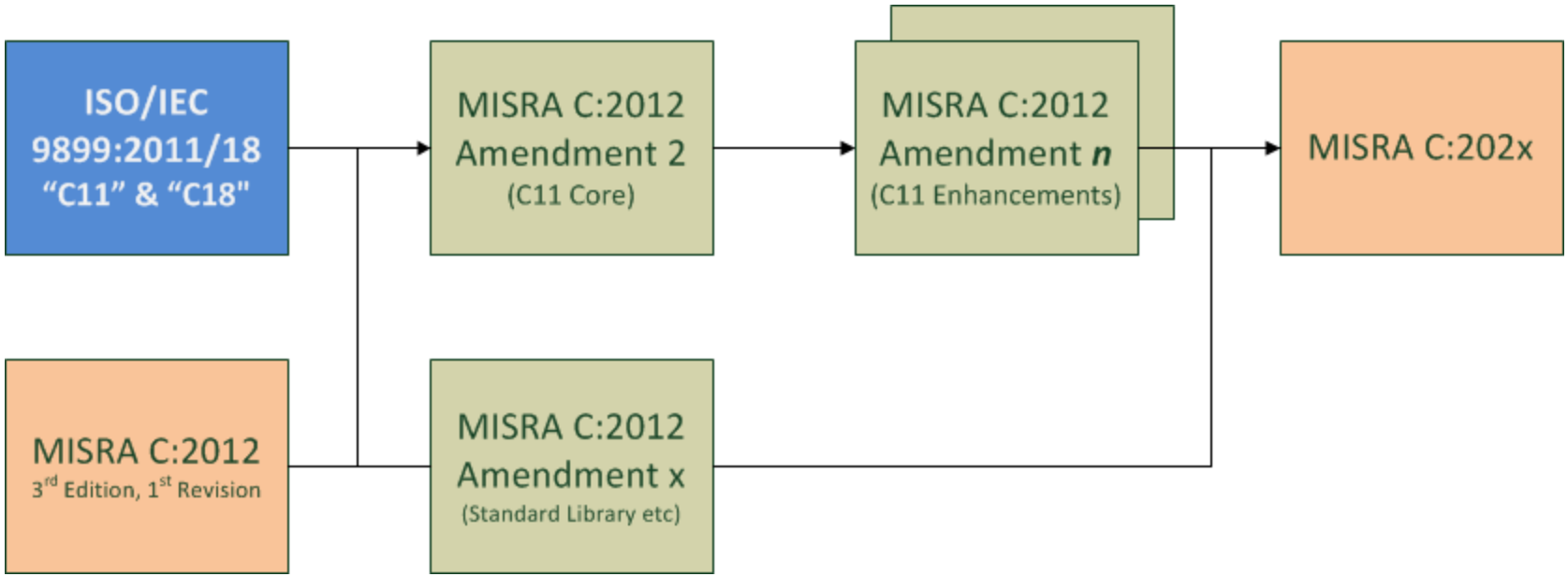
- MISRA C Working Group has commenced a review of:
  MISRA Cワーキンググループでは以下のレビューを開始
  - Changes with respect to the “core” functionality of C99
    C99の「コア」機能に関する変更
  - Atomic primitives　アトミックプリミティブ
  - Multi-threading　マルチスレッド
  - Unicode characters　Unicode文字
  - Appendix F/G – ISO/IEC 60559 floating point
    付録F/G – ISO/IEC 60559浮動小数点
  - Appendix K – bounds-checking functions　付録K – 境界チェック関数
  - Appendix L – Analyzability　付録L – 解析析可能性
- A series of incremental updates to MISRA C:2012 is proposed
  MISRA C: 2012への一連の増分更新を提案
- A review of the Standard Library is also underway, with a view of establishing a more focussed targeting of the problem areas and support for hosted applications.
  問題領域のより焦点を絞ったターゲット設定とホストアプリケーションのサポートを確立し、標準ライブラリのレビューも進行中

MISRA C:2012 Amendment 2 – C11 / C18

- Many features are permitted: 多くの機能が許可されている
  - Additional floating-point characteristic macros (float.h)
    追加の浮動小数点特性マクロ
  - Unicode characters and strings (uchar.h) Unicode文字と文字列
  - Static assertions 静的アサーション
  - Anonymous structures and unions * 匿名の構造体と共用体
  - Support for opening files for exclusive access
    排他的アクセスのためにファイルを開くサポート
  - *aligned_alloc* *
  - *at_quick_exit*, *quick_exit* *
  - *timespec_get* *

    * Subject to restrictions 制限の対象
- Some are not: 一部はそうではない
  - Type generic expressions 型ジェネリックな式
  - Support for multiple threads of execution (stdatomic.h, threads.h)
    複数スレッド実行のサポート
  - Alignment of objects (stdalign.h) オブジェクトの境界調整
  - Bounds-checking interfaces 範囲チェックインターフェイス
  - No-return functions リターンしない関数
- Published (Soon – TBC) 公開(まもなく– 要確認)

MISRA C – Future Plans



MISRA Compliance

Activity

MISRA Compliance – Activity

- April 2016 — Publication of MISRA Compliance:2016
  Publication of MISRA C:2004 Permits (Edition 1)
  … Thanks to IPA/SEC and JAMA for ideas and comments
  アイデアとコメントを寄せてくれたIPA/SECとJAMAに感謝

- October 2019 — Publication of MISRA Compliance:2019 …
  … No technical changes – a “House keeping” update
  技術的な変更なし –「ハウスキーピング」アップデート
  Development of MISRA C:2012 Permits (Edition 1)
  MISRA C:2012「逸脱許可」の開発（第1版）

MISRA Compliance – Feedback

- Projects are now using MISRA Compliance
  プロジェクトは現在、MISRAコンプライアンスを使用

- The 2019 update adds some MISRA-related software development process information so that it does not need to be maintained within future versions of MISRA C and MISRA C++
  2019年の更新では、MISRA CとMISRA C++の将来のバージョンで維持しなくても済むよう、MISRAに関連するソフトウェア開発プロセス情報をいくつか追加

Future Ideas

Compliance and Adopted Code

Background – Native and Adopted Code

Native Code ネイティブコード

- Source code developed within the scope of the project
  プロジェクトの範囲内で開発されたソースコード
- Can be changed as necessary to make it compliant
  コンプライアンスのため、必要に応じて変更できる

Adopted Code 流用コード

- Developed outside of the scope of the project
  プロジェクトの範囲外で開発
- May be supplied in source or binary forms
  ソース形式またはバイナリ形式で提供される場合がある
- Generally uses header files as an interface
  通常、ヘッダファイルをインターフェイスとして使用
- Cannot usually be changed to make compliant
  通常、コンプライアンスになるるように変更はできない
  - Note that non-compliant doesn't necessarily mean "bad"
    非コンプライアントは必ずしも「悪い」という意味ではないことに注意

Background – Deviation “Reasons”

MISRA Compliance acknowledges that deviations may be justifiable for reasons that include:
MISRAコンプライアンスでは、次のような理由で逸脱が正当化される場合があることを認めている

- When integrating MISRA compliant adopted code
  MISRAコンプライアントの流用コードを統合する場合
- When non-compliant adopted code is being used
  非コンプライアントの流用コードが使用されている場合

Background – Code may be non-compliant

- It was never written to be MISRA compliant
  MISRAに準拠するように書かれたことがない
- It is compliant with a different MISRA version
  異なるMISRAバージョンに準拠

Sources of non-compliance 非コンプライアンスの原因

- Header file content and use (objects, declarations, macro expansion, etc.)
  ヘッダファイルの内容と用途（オブジェクト、宣言、マクロ展開など）
- Interaction between modules that are compliant in isolation (e.g. identifier name clashes)
  独立してコンプライアントなモジュール間の相互作用（例：識別子名の衝突）

Problem

- How can Compliance be verified using static analysis without the tools generating lots of “noise” (sometimes called “false-positives”)?
  ツールが大量の「ノイズ」（「偽陽性」）を生成せずに、静的解析を使用してコンプライアンスを検証するには？

Solutions

- Some violations will have to be resolved as they relate to code behaviour issues:
  コードの動作の問題に関連するため、いくつかの違反は解決する必要あり
  - Identifier name collisions　識別子名の衝突
  - Violations of Mandatory Guidelines　必須ガイドラインの違反
- Many are simply related to using code that has not been written to comply with MISRA Guidelines
  多くは単に、MISRAガイドラインにコンプライアンスするように記述されていないコードの使用に関連するもの
  - It is assumed that this code is of appropriate quality and can be proven to work correctly, but…
    コードは適切な品質であり、正しく機能することが証明されていると想定されている、しかし、
  - How can this code be included within the MISRA checking process without the overhead involved in reviewing (potentially) many, many violations?
    （潜在的に）多数の「違反」のレビューに伴うオーバーヘッドなしで、このコードをMISRAチェックプロセスにどのように含めることができるか？

Function Contracts and Annotations
関数コントラクトとアノテーション

- One possible solutions that MISRA are considering is the use of function contracts supported by the use of annotations
  MISRAが検討している1つの可能な解決策は、アノテーションの使用によってサポートされる関数コントラクトの使用

- Function contracts allow:　関数コントラクトでは次のことができる
  - A set of pre-conditions to be defined that must be satisfied when making a call to a function
    関数を呼び出すときに満たすべきものとして定義される事前条件
  - A set of post-conditions to be defined that will be satisfied when a function returns
    関数が戻るときに満たすべきものとして定義される事後条件

MISRA will only introduce the use of contracts if they can be implemented in a way that the majority of C and C++ programmers can understand their use
MISRAはC、C++プログラマーの大部分がその使用を理解できる方法で実装できる場合にのみ、コントラクトの使用を導入する

How do function contracts help?
関数コントラクトの利点

- Providing function contracts for each of the API entry points defined within the header files of a piece of Adopted code allows its use to be validated
  流用コードのヘッダーファイルで定義される各APIエントリポイントに関数コントラクトを提供することで、その使用を検証できる

- There will be no need to perform complete MISRA checking on the Adopted code source files themselves
  流用コードのソースファイル自体に対して完全なMISRAチェックを実行する必要はない
  - Checks related to identifiers and Mandatory Guidelines would be required, but there should be no violations
    識別子と必須ガイドラインに関連するチェックは必要になるが、違反はないはず

- There is some additional work, as the contracts themselves will also need to be validated within the Adopted code
  コントラクト自体も流用コード内で検証する必要があるため、追加作業がある
  - This should be carried out by the “owner” of the Adopted code
    これは、流用コードの「所有者」によって実行されるべき

Annotations

- Annotations are used to help support the specification of function contracts, allowing:
  アノテーションは関数コントラクトの仕様をサポートするために使用され、以下を可能にする
  - (Raw) pointers and arrays to be differentiated
    （生）ポインタと配列を区別する
  - Better (though still not complete) detection of the invalid use of pointers and array bound violations
    ポインタの無効な使用と配列境界違反の（完全ではないが）より良い検出
  - Better enforcement of constraints on the values held by objects
    オブジェクトが保持する値に対する制約の強化
  - …

Pre-conditions and post-conditions

- *Pre-conditions* describe the constraints that the caller must satisfy when calling a function. For example:
  *事後条件*は、関数を呼び出すときに呼び出し元が満たさなければならない制約
  - Pointer must not be null
    ポインターはnullであってはならない
  - Parameter must point to a zero-terminated C string
    パラメータは、null終端するC文字列を指さねばならない
- *Post-conditions* describe conditions that the function guarantees hold when it returns. For example:
  *事後条件*は、関数が戻るときに保持することを保証する条件
  - Pointer may be null
    ポインタはnullでもよい
  - A zero-terminated C string will be returned
    null終端するC文字列が返される

Tools need to be “well behaved”

- A diagnostic is required when a pre- or post-condition cannot be proven to hold, including when this is due to checking undecidability
  事前または事後条件が成立することを証明できない（決定不能性のチェックによるものである場合を含む）場合、診断が必要
  - Which is generally the behaviour given by constraint solvers / theorem provers
    これは一般的に、制約ソルバーや定理証明者によって与えられた動作
- This could be seen as generating “false-positives”, but it generally indicates where extra checks are needed within the native code
  これは「偽陽性」の生成と見なすことができるが、通常ネイティブコード内で追加のチェックが必要な場所を示す
  - Consider a function having a pre-condition that a pointer parameter must not be null – a call to that function would need to be guarded with a null pointer check if a tool reported that it could not prove that the pointer was never null.
    ポインタパラメータがnullであってはならないという事前条件を持つ関数を考える。ツールでがポインターがnullでないことを証明できないと報告された場合、その関数の呼び出しはnullポインタチェックで保護する必要がある

Example – Standard Library strcpy()

- The *strcpy* function copies the C string from the array pointed to by s2 into the array pointed to by s1.
  *strcpy*関数は、s2が指す配列からs1が指す配列にC文字列をコピーする
  - The source string must include a terminating null character to avoid *undefined-behaviour*.
    *未定義動作*を避けるため、ソース文字列には終端のnull文字を含める必要がある
  - The destination buffer must be large enough to hold the string, including the zero terminator.
    コピー先のバッファは、null終端を含む文字列を保持するのに十分な大きさである必要がある
  - The copy includes the terminating null character.
    コピーには、終端のnull文字が含まれる
  - s1 is returned s1が返される
  - Others (pointers must not be null, …)
    その他（ポインタはnullであってはならない など）

Example – Defining the contract

- A *pre-condition* can be used to formally specify that “the source string must include a terminating null character to avoid *undefined-behaviour*”:
  *事前条件*を使用して、「未定義動作を回避するためソース文字列には終端null文字を含めなければならない」ことを正式に指定できる

```
char * array strcpy( char * array s1, const char * array s2 )
  pre( exists i in 0..s2.upb :- s2[ i ] == ( char )0 )
  ;
```

  - `array` and `pre` are macros with empty expansions
    `array`と`pre`は、空に展開されるマクロ
  - `s2.upb` gives the upper bound (length) of the source buffer (array)
    `s2.upb`は、ソースバッファ（配列）の上限（長さ）を示す

Example – Defining the contract

- A *post-condition* can be used to formally specify that "the copy includes the terminating null character":
  *事後条件*を使用し、「コピーに終端null文字が含まれる」ことを正式に指定できる

```
char * array strcpy( char * array s1, const char * array s2 )
  post( forall i in 0..strlen( s2 ) :- s1[ i ] == s2[ i ] )
  ;
```

  - `array` and `pre` are macros with empty expansions
    `array`と`pre`は、空に展開されるマクロ
  - `strlen( s2 )` given the index of the zero terminator within `s2`
    `s2`内のnull終端のインデックスを指定する `strlen( s2 )`
  - Note that `strlen( s2 )` may be less than `s2.upb`
    `strlen( s2 )`は`s2.upb`より小さい場合があることに注意

Example – Defining the contract

- Multiple pre- and post-conditions can be specified:
  複数の事前、事後条件を指定できる

```
char * array strcpy( char * array s1, const char * array s2 )
  pre ( exists i in 0..s2.upb :- s2[ i ] == ( char )0 )
  /* More pre conditions ... */
  post( forall i in 0..strlen( s2 ) :- s1[ i ] == s2[ i ] )
  post( result == s2 )
  ;
```

  - `result` is used to specify the value returned by the function
    `result`は関数が返す値を指定するために使用

- Tools may be used to automatically detect implicit pre- and post-conditions
  暗黙の事前、事後条件を自動的に検出するツールを使用できる

  - For example, unguarded use of a pointer parameter within a function implies a pre-condition stating that the pointer must not be null.
    たとえば、関数内でのポインタパラメータの無防備な使用は、ポインタがnullであってはならないことを示す事前条件を意味する

  - This could possibly used to retrospectively add information to adopted code when the full source is available.
    完全なソースが利用可能な場合、流用コードに遡及的に情報を追加するために使用される可能性がある

MISRA Compliance with function contracts

Native code

- Undergoes conventional MISRA checking 従来のMISRAチェックを受ける
- Function contracts and annotations are used
  関数コントラクトとアノテーションを使用することで、
  - Optional within bulk of code, but are likely to improve the quality of static analysis
    大量のコード内で静的解析の品質を向上させる可能性がある
  - To ensure that relevant information is available when calling Adopted code
    流用コードを呼び出すときに関連情報が利用できることを保証する

Adopted Code

- Does not need to undergo MISRA checking MISRAチェックは不要
- Function contracts must be validated
  関数コントラクトを検証する必要がある

MISRA Compliance with function contracts

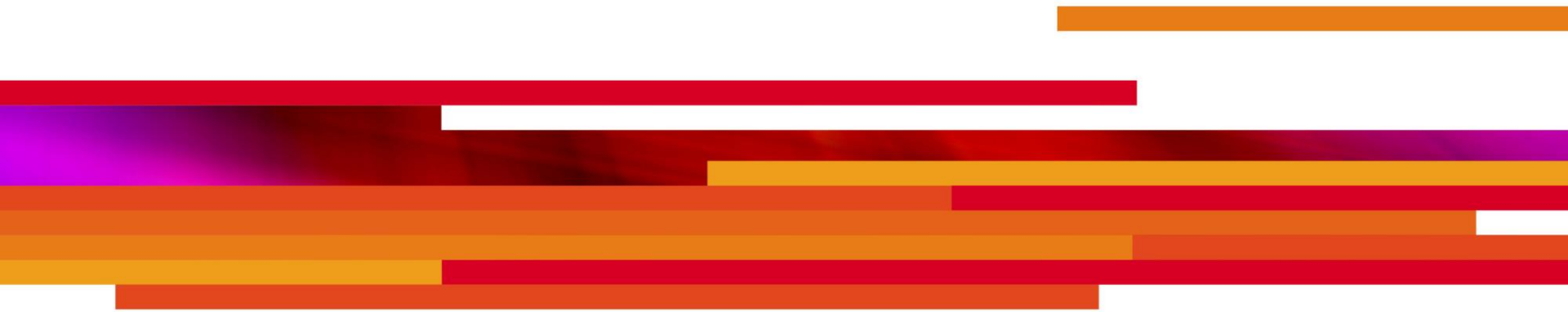
- The brief introduction given in the example shows that it is relatively easy to apply function contracts to the Standard Library
  例は、標準ライブラリに関数コントラクトを適用するのが比較的簡単であることを示す
  - The behaviour is defined by an international standard
    動作は国際規格で定義
  - This means the pre- and post-conditions are formally specified and can therefore be unambiguously checked
    事前、事後条件が正式に指定されているため、明確にチェックできることを意味する
- The same is also possible with adopted code having a well-defined set of requirements
  同様に、明確に定義された一連の要件を持つ流用コードでも可能
  - Which means MISRA will still not make it easy to use low-quality adopted code within a MISRA context!
    つまり、MISRAを使用してもMISRAのコンテキストで低品質な流用コードを使用することは簡単ではない！

Summary

Summary

- The evolution of MISRA C and C++ continues …
  MISRA CおよびC++の進化は続く
- Both plan a much high release cadence
  両者とも非常に高いリリース周期を計画
- Cooperation between the groups will enable the use of language-agnostic guidance documents to be produced with the groups adding any necessary language-specific details
  グループ間の協力で言語に依存しないガイダンス文書の使用が可能になり、グループは必要な言語固有の詳細を追加する
- MISRA is starting to research topics that go beyond language subset development, with the aim of improving the precision and recall of static analysis tools
  MISRAでは静的解析ツールの精度とリコールを改善することを目的とし、言語サブセットの開発を超えたトピックの研究を開始
  - Function contracts　機能契約
  - Supporting the drafting of an ISO/IEC standard for static analysis
    静的解析のためのISO/IEC規格の起草をサポート

Get Involved...



MISRA C and C++

- The MISRA C and MISRA C++ Working Groups are made up of volunteers.
  MISRA C、MISRA C++ワーキンググループはボランティアで構成

- Membership is open to anyone with the appropriate technical ability.
  メンバーシップは、適切な技術力を持つ人なら誰でも利用可能

- Anyone interested in joining should contact the respective Working Group Chairman:
  参加に興味ある方は、それぞれのワーキンググループの議長にご連絡ください
  - For MISRA C chair-c@misra.org.uk
  - For MISRA C++ chair-cpp@misra.org.uk

Contact Us

FUJI SETSUBI

Need more information?

LDRA.com

LinkedIn

info@ldra.com

富士設備工業(株)電子機器事業部 https://www.fuji-setsu.co.jp/