IPA Better Life with IT

Embedded Technology 2017 IoT Technology 2017

組込みソフトウェア開発向けコーディング作法ガイド ESCR［C言語版］の改訂について

～ セキュアコーディングへの対応 ～

2017年11月15日・16日

独立行政法人 情報処理推進機構（IPA）
技術本部 ソフトウェア高信頼化センター（SEC）
十山 圭介

目 次

ESCRとは

SEC
Software Reliability Enhancement Center

- 組込みシステムの信頼性向上をミッションとして、2004年に開始されたIPA/SEC活動の成果物として作成

目的　　：実用的なコーディング規約の策定・運用の促進
対象言語：C言語、C++言語
対象読者：規約の作成者、プログラマー、レビューアー

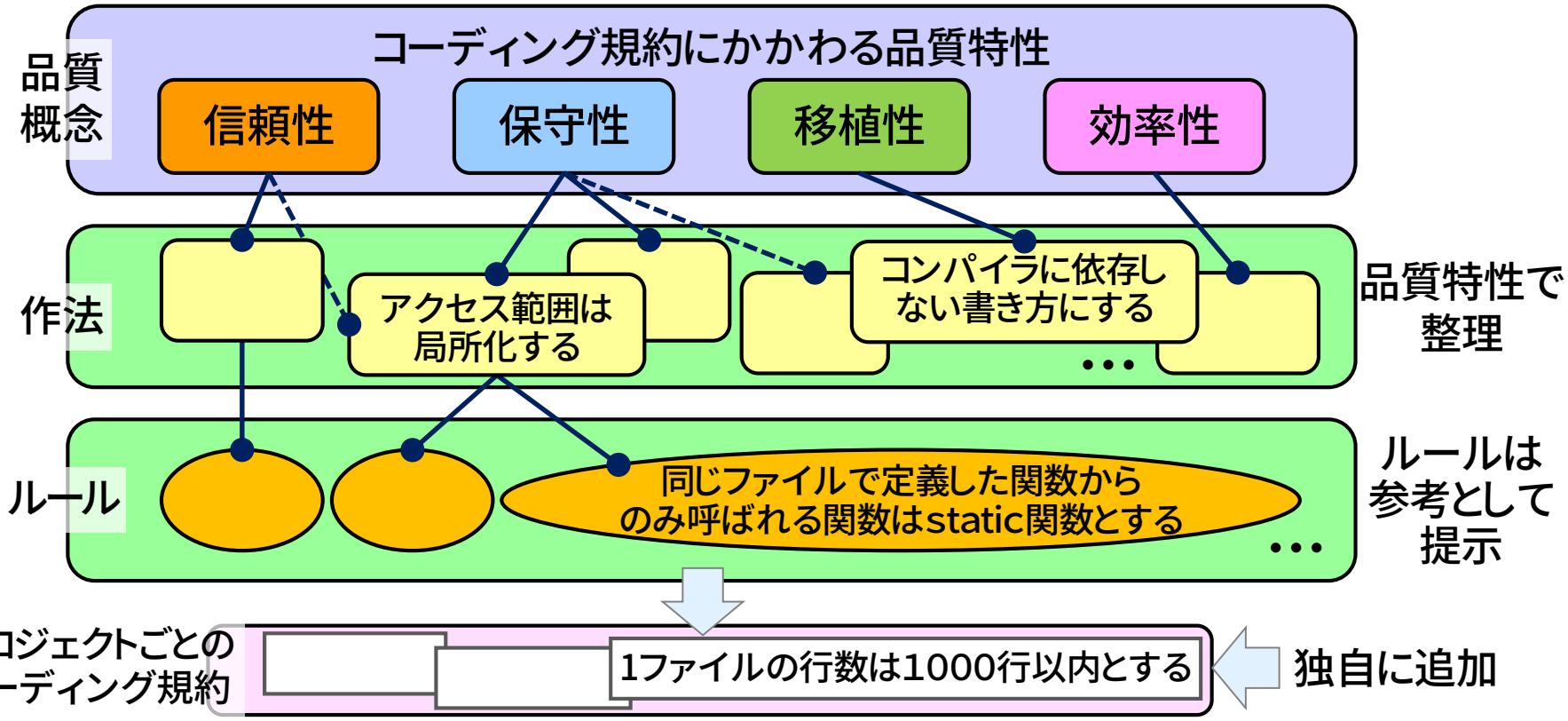
- **ESCRの特徴**
  - 品質特性により、ルールをトップダウンに体系化
    - 信頼性、保守性、移植性、効率性 といった品質特性で整理し、理解促進
  - すぐに使えるルールのリファレンス
    - 作法に対応し、MISRAや、GNU、参加企業のコーディングルールなどを提示
  - ルールの特性を提示
    - 対応する品質特性を保つために重要なルール、など
  - 初級者にもわかり易い表現

コーディング規約を分かりやすくするために SEC

Software Reliability Enhancement Center

- ソフトウェアの品質特性をもとに、作法とルールを体系化する

作　法：品質向上のために守るべき具体的な実装の考え方
ルール：言語依存性を考慮した具体的なコーディングの決め事



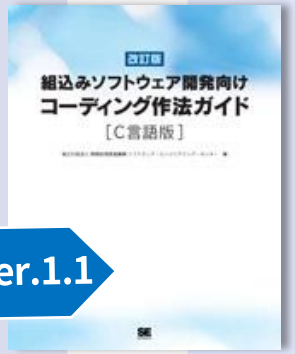
ESCR策定の沿革

SEC
Software Reliability Enhancement Center

’04 ’05 ’06 ’07 ’08 ’09 ’10 ’11 ’12 ’13 ’14 ’15 ’16 ’17

コーディング作法ガイド WG活動開始

ESCR C 作成開始

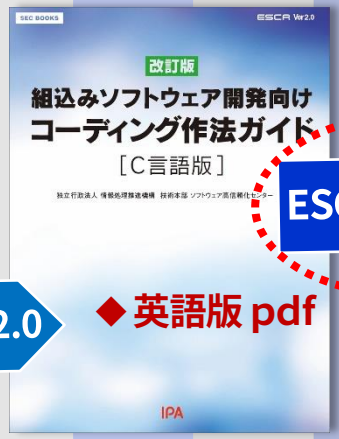


ESCR C Ver.1.0

ESCR C Ver.1.1

ESCRのJIS化（JIS X 0180）

ESCR C 改訂開始



ESCR C Ver.3.0

ESCR C Ver.2.0

◆英語版 pdf

ESCR C++ Ver.1.0

ESCR C++ 対応開始

ESCR C++ Ver.2.0

ESCR C++ 改訂開始

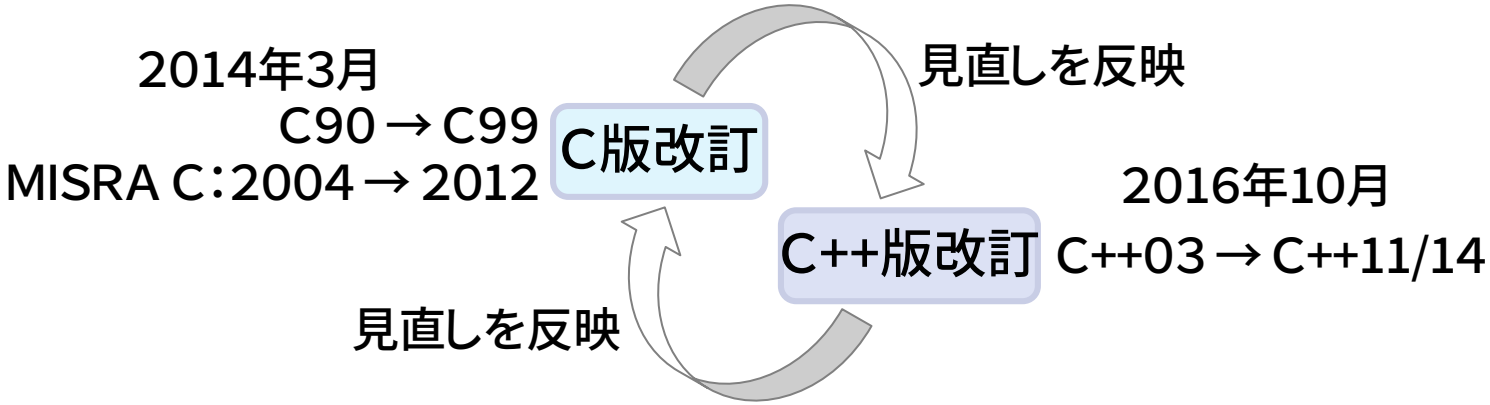
◆英語版 pdf

- 品質と生産性の向上に有用との評価
- 書籍および pdf のダウンロード 3万部以上

ESCRの改訂成果

SEC
Software Reliability Enhancement Center

準拠する言語規格など



セキュリティへの対応（品質規格改訂への対応）

- JIS X 0129の後継である25010でセキュリティが特性に格上げされた
  → 説明を追加するが、セキュリティ特性は採用せず
  セキュリティは、基本的には設計段階の特性と考え、バッファオーバーフローなどセキュリティに影響するコーディングもある　とする
- 脆弱性との対応付け、CERT Cコーディングスタンダードとの対応付けを明確化する

3．セキュアコーディングに向けて

ESCRでのセキュリティ対応の強化

SEC Software Reliability Enhancement Center

- 現状のESCRのルールにはセキュリティの観点から重要なものが含まれる
  - それらとCERT C コーディングスタンダードのルールとの対応付けを行い、コーディング規約の作成段階からセキュリティを念頭に置くことの重要性を示す

CERT C ルール		ESCR ルール
EXP34-C	nullポインタを参照しない	R3.2.2 ポインタは、ナルポインタでないことを確認してからポインタの指す先を参照する
INT33-C	除算および剰余演算がゼロ除算エラーを引き起こさないことを保証する	R3.2.1 除算や剰余算の右辺式は、0 でないことを確認してから演算を行う

※ CERT C コーディングスタンダード
脆弱性に繋がる恐れのあるコーディング作法や未定義の動作を極力減らすことを目的にまとめられたコーディング規約。C言語を使ってセキュアコーディングを行うためのルールとレコメンデーションを定めたもの　https://www.jpcert.or.jp/sc-rules/

- ✓ 攻撃可能な脆弱性を作り込まない
- ✓ コードの保守性、移植性の向上

ESCR［C言語版］の改訂 ～ Ver.3.0に向けて

SEC Software Reliability Enhancement Center

- CERT Cルールの本編への取り込みの検討、
  IPAセキュリティセンター提案ルールの取込み、の検討
  → 54のルールや作法をリストアップして検討
  - セキュリティの考慮が不足したコーディングに関する注意点
    - バッファオーバーフローによる脆弱性
    - 整数オーバーフローによる脆弱性
    - ディレクトリトラバーサルによる脆弱性
    - パスワード処理などによる脆弱性
    - 同期処理での脆弱性
    - ・・・

- CERT Cルールの本編への取り込みの検討、
  IPAセキュリティセンター提案ルールの取込み、の検討
  → 54のルールや作法をリストアップして検討

  - 対象としたCERT Cルールの例（48項目）

ルール番号	内　容
PRE11-C	単一文からなるマクロ定義をセミコロンで終端しない
DCL05-C	typedef を使いコードの可読性を改善する
EXP09-C	型や変数のサイズは sizeof を使って求める
EXP13-C	関係演算子および等価演算子は、結合則が成り立たないものとして扱う
INT04-C	信頼できない入力源から取得した整数値は制限する
ARR01-C	配列のサイズを求めるときに sizeof 演算子をポインタに適用しない
STR08-C	新たに開発する文字列処理するコードには managed string を使用する
STR31-C	文字データと null 終端文字を格納するために十分な領域を確保する
MEM01-C	free() した直後のポインタには新しい値を代入する
・・・	

- IPAセキュリティセンター提案ルール（6項目）

項番	内　容
1	ポインタ変数に対してsizeof 関数を使用しない
2	ジャンクションを考慮し、無限ループにならないような処理にする
3	【作法詳細】入力値にプログラム上の特殊な意味を持つ文字が含まれる可能性を考慮する 【ルール】　入力値が特殊な意味として解釈されない関数を利用する
4	【作法詳細】ファイルやディレクトリに関する操作ではアクセス範囲を意識した書き方にする。 【ルール】　ディレクトリの指定は、それ以外のディレクトリにアクセスされないような記述をする
5	【作法詳細】メモリ管理は適切に行う 【ルール】　確保したメモリは確実に解放する
6	【作法詳細】一連の処理を複数のプロセスで実施しない 【ルール】　共有リソースに対して処理を行う際には排他制御を行う

検討結果 ～ Ver.3.0改訂項目

SEC Software Reliability Enhancement Center

- セキュアコーディング対応ルール、解説の追加

例：ルールの新設

- CERT C：配列のサイズを求めるときに sizeof 演算子をポインタに適用しない
- 「ポインタ変数に対して sizeof 演算を使用しない」（セキュリティセンター提案）

CERT C ARR01-C説明・違反コードより

```
enum {ARR_LEN = 100};

void clear(int a[ARR_LEN]) {
  memset(a, 0, sizeof(a)); /* 間違い */
}

int main(void) {
  int b[ARR_LEN];
  clear(b);
  assert(b[ARR_LEN / 2]==0); /* おそらく失敗 */
  return 0;
}
```

100*sizeof(int) と等しくならない。sizeof 演算子は、配列型または関数型として宣言された引数に適用されると、たとえ引数宣言で長さが指定されていても、型調整された(ポインタ)型のサイズを求めるからである。

len*sizeof(int) とする

以下でご紹介するルールや解説などは最終版のものではなく、
簡略化しています。

新ルールの例(1)

SEC
Software Reliability Enhancement Center

sizeof演算子

R3.1.5	(1) sizeof演算子はポインタ型の変数に用いてはならない。 (2) sizeof演算子は配列型の引数に用いてはならない。

sizeof(配列型の変数)は配列のサイズとなることから、sizeof(ポインタ型の変数)もポインタが指す領域のサイズになると勘違いすることがある。実際はポインタのサイズとなる。

・・・

【より詳しく知りたい人への参考文献】

CERT ARR01-C　CWE-467　MISRA C:2012 Amendment 12.5

参考となる文献を複数掲載

例

```
void func1(char *cp) {
  size_t x;
  x = sizeof(cp);  // NG cpはポインタ型の変数(引数)
```

新ルールの例(2)

SEC Software Reliability Enhancement Center

- 【作法】共有リソースに対して処理を行う際には排他制御を行う。

R3.11.1 並行処理ではvolatileを同期プリミティブとして使用しない。

並行処理や非同期的なシグナル処理では、データの更新結果を他のスレッドに適切に反映する必要がある。他のスレッドによる更新の不可分性を保証するための目的でvolatileが利用されていることがあるがこれは誤りである。

・・・

なお、同期用プリミティブの取得と解放は、同じ翻訳単位内の同一抽象レベルで行うことが望ましい。

例

```
int v = 0;
mtx_t flag; // 排他制御用ミューテックス
 ...
mtx_lock(&flag);
v++; // クリティカルセクション内。不可分に処理される
mtx_unlock(&flag);
 ...
```

修正の例(1)・・・解説追加

SEC
Software Reliability Enhancement Center

R3.2.2	ポインタは、ナルポインタでないことを確認してからポインタの指すメモリを参照する。

ナルポインタや適正でないメモリを指すポインタを介してメモリにアクセスするとハードウェアトラップやメモリ破壊が発生するため、対策が必要である。

対策の例:

(1)使用済みのポインタにナルポインタを代入する。ポインタの指す先を参照する前に検査することをルール化することで、メモリの解放後使用や二重解放が回避できる。

(2)近年の組込みOSにはポインタの値の適正を検査するシステムサービスを提供するものがある。これらのOSを使用する場合は、このシステムサービスを必ず使用する。ポインタによるメモリアクセスの前にポインタの値が適正であることを確認することで、不当なメモリへのアクセスを回避できる。

修正の例（2）・・・ 解説追加

SEC
Software Reliability Enhancement Center

M4.4.1	《ヘッダファイルに記述する内容（宣言、定義など）とその記述順序を規定する。》

・・・ ヘッダファイルには、複数のソースファイルで共通に使用するマクロの定義、構造体 ・・・ を記述する。

例えば、以下の順序で記述する。

（1）ファイルヘッダコメント

・・・

なお、typedef やマクロを利用することで、プログラムを一目でわかりやすくしたり、変更箇所を局所化したりできるが、規律なく利用すると同じ内容の別のマクロが定義されるなど、逆効果にもなる。

プロジェクトで利用するマクロや typedef は、プロジェクト開始時に規定し、一箇所で定義して統一的に利用するとよい。

コラムによる解説

SEC
Software Reliability
Enhancement Center

- ESCRのルールにはしないが重要な項目であるため、「コラム」を設けて解説する
  - 設計に近いと考えられる項目、ライブラリに関連する項目 など
    - 文字列ライブラリの使用、
    - 機密情報の取り扱い、
    - 信頼できない可能性のあるデータの取り扱い
      - 整数値
      - 書式文字列
      - ファイル名やパス名
      - ファイルの特定

    に分けて概説する
  - 対応するCERT Cルールを提示する

コラム概要

SEC
Software Reliability
Enhancement Center

- **文字列ライブラリの使用**
  ライブラリの多くの文字列操作関数には長さを検査する機能が組み込まれておらず、その使用がバッファオーバーフローの原因となり得る
  - 古い規格の文字列処理関数は使用を避けるようにするとともに、文字列操作には境界チェックインターフェースを使用する
  - C11では標準文字列処理関数の代替関数を規定
    CERT Cでは領域あふれが生じにくい文字列操作ライブラリの使用を推奨
- **機密情報の取り扱い**
- **信頼できない可能性のあるデータの取り扱い**
  - **整数値**
  - **書式文字列**
    - 入力をそのまま書式文字列として処理することのないよう注意する
  - ファイル関連

信頼できない可能性のあるデータの取り扱い

- **整数値**
  その上限と下限を特定できるかどうかを確認するようにする（INT04-C）
- **書式文字列**
  printf 関数などの引数として外部から制御可能な書式文字列が与えられると、バッファオーバーフローを招く可能性がある。「%n」は・・・
  - 入力をそのまま書式文字列として処理することのないよう注意する（FIO30-C）
  - 全ての書式文字列関数が、ユーザが制御できない静的な文字列であること、さらに、その関数に適切な数の引数が渡されていることを確認する（FIO47-C）
  - 可能であれば、書式文字列において「%n」をサポートしない関数を使用する（C11では「%n」書式が廃止）
- **ファイル名やパス名**
- **ファイルの特定**

4．今後の活動

今後の活動

SEC
Software Reliability Enhancement Center

- 改訂版（Ver.3.0）の公開は 2018年1月を予定
- CERT C++ と ESCR C++言語版のルール対応表
  2018年3月公開を予定
- MISRA C++ 改版への対応
  - MISRAの改版時期に合わせて実施（2018年度以降の見込み）

ご清聴ありがとうございました

今後とも ESCR C/C++ のご活用をよろしくお願いします

Also available in English


IPA
Better Life
with IT